Parameter Name	*Form*	*Description*	*Where Used*
PDEAttribute	Enumeration	Choice of available *PDE* attributes. The following values are allowed: “name” “gid” “groupName” “description” “type” “executable” “maxAntecedents” “createDate” “createNode” “author” “userInfo” “supplierInfo” “checksum”	*getPDEdirectory*(), *PDEAttributeItem* parameter
PDEAttributeItem	(*PDEAttribute*, *PDEAttributeValue*)	Contains the enumerated identifier of a particular *PDE* attribute and the value of that attribute.	*PDEDirItem* parameter
PDEAttributeName	Enumeration	Choice of available *PDE* attributes. The following values are allowed: “name” “gid” “groupName” “description” “type” “executable” “createDate” “createNode” “author” “userInfo” — supplied value matched to any *userInfo* string in the *PDE* “supplierInfo” — supplied value matched to any *supplierInfo* string in the *PDE*	*PDEFilter* parameter
PDEAttributeValue	Depends on selected attribute	Contains the value of the corresponding attribute in the appropriate format.	*PDEAttributeItem* parameter, *PDEFilter* parameter
PDEDirItem	(*uid*, list of *PDEAttributeItem*)	Single directory entry returned by *getPDEdirectory*().	*getPDEdirectory*()
PDEFilter	(*PDEAttributeName*, operator, *PDEAttributeValue*)	Set of values that define a filter on the set of *PDEs* for the *getPDEdirectory*() service.	*getPDEdirectory*()
pdeRef	UUID	Contains a reference to a *PDE*. The reference is a UUID value that may represent a *gid* or a *uid*.	*inputMap* parameter, *outputMap* parameter
resolution	UUID	Contains the value of the *uid* attribute of the *PDE* to which a *PDE* reference is resolved.	*inputMap* parameter, *outputMap* parameter
resPDEinfo	list of (*pdeRef*, *resPDEstat*)	Contains a list of all references to *PDEs* found in the recipe structure with its corresponding status.	*resolvePDE*()

Parameter Name	*Form*	*Description*	*Where Used*
resPDEstat	Error	Status response for the *resolvePDE*() service. The following responses are allowed: "OK" — no problems were encountered "MissingTargetPDE" — the *targetPDE* value could not be resolved to any *PDE* on the *EquipmentNode.* "MissingMapPDE" — A needed PDE specified by the *inputMap* could not be found on the *EquipmentNode.* "MissingReferencedPDE" — A needed *PDE* could not be found on the *EquipmentNode*. If more than one of these conditions applies, the first one on this list that applies should be reported.	*resPDEinfo* parameter
rtsRspStat	Enumeration	Status response for the *requestToSendPDE*() service. The following responses are allowed: "OK" — Indicates permission to send is granted. "NoResources" — Indicates there are not sufficient resources available to service this request (for example, disk space). "Other" — Unspecified error occurred.	*requestToSendPDE*()
sendRspInfo	list of (*uid, sendRspStat, verifyRspStat*)	Contains a list of all problems encountered with the *sendPDE*() request. The uid value indicates which specified *PDE* caused the error.	*sendPDE*()
sendRspStat	Error	Status response for the *sendPDE*() service. The following responses are allowed: "OK" — Indicates *PDE* was successfully received. "NoResources" - Indicates there are not sufficient resources available to receive this *PDE* (for example, disk space). "TargetMismatch" — The receiver is not compatible with the *PDE's ExecutionTarget*. "PDElocked" – The PDE already exists and cannot be replaced at this time. "VerificationFailed" — The *PDE* failed verification. See *verifyRspStat* for failure reason. "Other" — A problem occurred that is not described by the other enumerated values.	*sendRspInfo* parameter
targetPDE	UUID	The *uid* attribute value of the *PDE* to be acted upon.	*verifyPDE*(), *resolvePDE*()
tcid	UUID	Identifier of a *TransferContainer*. This is a uuid value.	*getPDE*(), *getPDEheader*(), *requestToSendPDE*(), *sendPDE*()
transferContainer	Binary	Container format is left as an implementation detail.	*getPDE*(), *getPDEheader*(), *sendPDE*()
transferSize	Integer	Size in bytes of the *TransferContainer* to be transferred.	*requestToSendPDE*()
uid	UUID	Refers to a *PDE* by its *uid* attribute.	*deletePDE*(), *getPDEheader*(), *getPDE*(), *getRspInfo* parameter, *delRspInfo* parameter *PDEDirItem* parameter, *verifyInfo* parameter

Parameter Name	*Form*	*Description*	*Where Used*
verifyDepth	Enumeration	Verifies the scope of the *PDEs* to be verified. "Single" — Only the specified *PDE* is checked. "All" — Verify the specified *PDE*, plus all *PDEs* it references directly or indirectly (e.g. by the *PDEs* that are referenced). It requires a check that all required *PDEparameters* are supplied by each referencing *PDE*. This may be used with a Master *PDE* to verify all the *PDEs* it will need for an activity. Note that if a PDE is referenced by multiple other *PDEs* within the scope of a *verifyPDE*() request, it only needs to be verified once during this process.	*verifyPDE*()
verifyInfo	list of (*uid*, *verifyRspStat*)	One value set is returned for each *PDE* that was reviewed by the verification process. The *uid* identifies the *PDE* that was found to have an error.	*verifyPDE*()
verifyRspStat	Error	Status response for the *getPDE*() service. The following values are allowed: "OK" — Indicates successful verification of the *PDE.* "NotFound" — The specified *PDE* was not found. "ChecksumFail" — The computed checksum of the *PDEheader* or of the *PDEbody* did not match the values in the *PDE*. "SyntaxError" — The syntax of the *PDE* or its body was found to be incorrect. "ContentError" — The content of the *PDE* did not match allowed values or did not follow all content rules. "Other" — The verification error did not match the other enumerated categories.	*verifyInfo* Parameter, *sendRspStat* Parameter
verifySuccess	Boolean	Returns "true" if no errors were encountered during the entire *verifyPDE*() operation. Otherwise returns "false".	
verifyType	Enumeration	Specifies the type of verification to perform. The following values are allowed: "Checksum" — computes the checksum of each applicable *PDE* and compares the value to the *checksum* value contained in the *PDE*. It computes the checksum of any related external *PDEbodies* and compares the value to the *bodyChecksum* value contained in the *PDE.* It also confirms that the *PDEbody* is matched to the proper *PDE*. "Validity" – For each *PDE* to be verified, ensures that the *PDE* can be executed as part of a recipe without error. This verification checks the syntax of the *PDE*. It includes "Checksum" verification plus any other criteria the equipment can provide. Also included is a check that a compatible ExecutionTarget exists on the equipment (if any ExecutionTargets are specified for this PDE). Successful verification does not guarantee the desired process results.	*verifyPDE*()

8.4.2.4 *Service Message Definitions*

8.4.2.4.1 This section specifies the allowable/required parameters for each service. Table 17 shows the format of the Service Parameters tables used for this purpose.

Table 17 Service Parameters

Parameter Name	*Req/Ind*	*Rsp/Cnf*	*Comment*
Name	see below	see below	Descriptive Text

8.4.2.4.2 Each service has an initial message (Request to the sender, Indicator to the receiver, hence Req/Ind). Services that are "Request/Response" have a return message (Response to the sender, Confirmation to the receiver, hence Rsp/Cnf). The Req/Ind and Rsp/Cnf columns indicate in which message each parameter is to be included. The codes from Table 18 are used in these columns to give further information.

Table 18 Req/Ind and Rsp/Cnf Codes

Code	*Description*
M	Mandatory Parameter — Must be given a valid value.
C	Conditional Parameter — May be defined in some circumstances and undefined in others. Whether a value is given may be completely optional or may depend on the value of the other parameters.
U	User-Defined Parameter.
-	The parameter is not used.
=	(For response only) Indicates that the value of this parameter in the response must match that in the primary (if defined).

8.4.2.5 *getPDEdirectory*()

8.4.2.5.1 The *getPDEdirectory*() service is used by the FICS to request a list of *PDEs* maintained by the service provider. The requestor may supply a filter that will yield a subset of the full list of *PDEs*. A subset of the PDEheader's attributes can be reported for each returned *PDE* if desired.

8.4.2.5.2 When multiple *PDEFilters* are specified, these filters are to be AND'ed together. That is, each filter is applied in turn with the reduced list that is output from one becoming the input for the next. Note that no PDEDirItems are returned when no PDE's are found that pass the filters.

8.4.2.5.3 The requestor may choose not to include PDEFilters or PDEAttributes in the request.

Table 19 *getPDEdirectory*() Service Parameters

Parameter Name	*Req/Ind*	*Rsp/Cnf*	*Comment*
list of PDEFilter	C	-	List of criteria for which *PDEs* to report. If no *PDEFilters* supplied, all *PDEs* will be reported
list of PDEAttribute	C	-	Lists the attributes to be returned for each *PDE* reported
list of PDEDirItem	-	C	List of the *PDEs* on the service provider that match the provided filter. The values of the specified attributes are included for each.
dirRspStat		M	Denotes any error found. If there is an error, no *PDEDirItems* are returned.

8.4.2.6 *deletePDE*()

8.4.2.6.1 The *deletePDE*() service is used by the FICS to request that specified *PDEs* be deleted from the service provider. The request includes a list of the *PDEs* to be deleted. The requestor may wish to first use the *getPDEdirectory*() service to obtain a list of the available *PDEs*.

8.4.2.6.2 The *delRspInfo* lists any errors encountered by the service provider in the completion of the *deletePDE*() request. All *PDEs* not referenced by *delRspInfo* shall be deleted.

8.4.2.6.3 If the list of *uid* contains multiple references to the same *PDE*, the *RapNode* shall delete the *PDE* and provide a successful response.

Table 20 *deletePDE*() Service Parameters

Parameter Name	*Req/Ind*	*Rsp/Cnf*	*Comment*
list of *uid*	M	-	Specifies which *PDEs* are to be deleted. At least one *PDE* must be listed.
delRspInfo	-	M	Information concerning the result of the service.

8.4.2.6.4 RaP allows deletion of a *PDE*, even if it is referenced by another *PDE*. The user should be aware that deletion of *PDEs* may make execution of one or more *Master PDEs* impossible due to unsatisfied references.

8.4.2.7 *getPDEheader*()

8.4.2.7.1 The *getPDEheader*() service is used to request that the headers of one or more *PDEs* be returned from the service provider. Prior to sending a response, the service provider shall be responsible for locating the *PDEs* requested, extracting the headers, packaging them in a *TransferContainer*, and generating a tcid for the *TransferContainer*.

8.4.2.7.2 When reported separately from its *PDEbody*, a *PDEheader* serves only a documentation role and is not usable as part of a recipe. The *PDEbody* is never transferred separately from its *PDEheader*.

8.4.2.7.3 The *getRespInfo* parameter contains information about any errors that occur. If errors are encountered, then the *TransferContainer* shall contain all requested *PDEheader*s for which no error is reported.

8.4.2.7.4 If the list of *uids* contains duplicate entries, the *RapNode* shall combine these requests and (in the absence of other problems) return only one copy of the requested *PDEheader*. The returned *getRspStat* value for each original request shall reflect the status of the combined request (that is, "OK" if one copy is returned).

Table 21 *getPDEheader*() Service Parameters

Parameter Name	*Req/Ind*	*Rsp/Cnf*	*Comment*
list of *uid*	M	-	Indicates the identifiers for the *PDEs* whose headers are to be returned in the *TransferContainer*.
tcid	-	C	uuid of the *TransferContainer* that has been returned.
transferContainer	-	C	The actual *TransferContainer*. It is included only if *PDEheaders* are returned as a result of the request.
getRspInfo	-	M	Report success or failure for each requested *PDEheader*.

8.4.2.8 *getPDE*()

8.4.2.8.1 The *getPDE*() service is used to request that one or more *PDEs* be returned from the service provider. Prior to sending a response, the service provider shall be responsible for locating the *PDEs* requested, packaging them in a *TransferContainer*, and generating a uuid for the *TransferContainer*.

8.4.2.8.2 The *getRespInfo* parameter contains information about any errors that occur. If errors are encountered, then the *TransferContainer* shall contain all requested *PDEs* for which no error is reported.

8.4.2.8.3 If the list of *uids* contains duplicate entries, the *RapNode* shall combine these requests and (in the absence of other problems) return only one copy of the requested *PDE*." The returned *getRspStat* value for each original request shall reflect the status of the combined request (that is, "OK" if one copy is returned).

Table 22 *getPDE*() Service Parameters

Parameter Name	*Req/Ind*	*Rsp/Cnf*	*Comment*
list of *uid*	M	-	Indicates the identifiers for the *PDEs* requested to be returned in the *TransferContainer*.
tcid	-	C	uuid of the *TransferContainer* that has been returned.

transferContainer	-	C	The actual *TransferContainer*. It is included only if *PDEs* are returned as a result of the request.
getRspInfo	-	M	Report success or failure for each requested *PDE*.

8.4.2.9 *requestToSendPDE*()

8.4.2.9.1 Request for permission to send a *TransferContainer* containing *PDEs* to the service provider. The size of the proposed *TransferContainer* is provided so that the service provider can ensure that room is available.

8.4.2.9.2 The response to the request is contained in the *rtsRspStat*. If the returned value is "OK", the service provider is indicating that the proposed *sendPDE*() service should succeed. The *requestToSendPDE*() service shall not be required to be send prior to the *sendPDE*() service. It provides an opportunity to check on a proposed transfer before that transfer is attempted.

8.4.2.9.3 The tcid appears in both the requestToSendPDE() and sendPDE() services. This provides a way for the RaPnode to match a received TransferContainer with the earlier requestToSendPDE() message.

Table 23 *requestToSendPDE*() Service Parameters

Parameter Name	*Req/Ind*	*Rsp/Cnf*	*Comment*
tcid	M	=	Identification of the TransferContainer to be transferred.
transferSize	M	-	Size of the TransferContainer in bytes.
rtsRspStat	-	M	Response to the request to send.

8.4.2.10 *send*PDE()

8.4.2.10.1 The *sendPDE*() service is used to transfer *PDEs* contained in a *TransferContainer* to the service provider. Any response other than success ("OK") shall be considered an indication that the recipe was not accepted. If a duplicate *PDE* is contained in the *TransferContainer*, the service provider shall store one and discard the other. This shall not be considered an error. If a *PDE* in the *TransferContainer* is already stored on the service provider, the newly received *PDE* shall overwrite the existing one.

8.4.2.10.2 Refer to ¶8.5.1.3 for requirements related to PDE transfer that apply to EquipmentNodes.

Table 24 *sendPDE*() Service Parameters

Parameter Name	*Req/Ind*	*Rsp/Cnf*	*Comment*
tcid	M	-	Identifier of the TransferContainer being sent.
transferContainer	M	-	The actual TransferContainer being transferred.
sendRspInfo	-	M	Report success or failure for each PDE.

8.4.2.11 *resolvePDE*()

8.4.2.11.1 In the recipe hierarchy, each reference of a *PDE* is either a *uid* or a *gid* value. All *gid*s must be resolved to *uid*s before a recipe can be executed. The *EquipmentNode* is responsible for the final resolution of any *gid*s not resolved by the client (see ¶8.5.4). The *resolvePDE*() service is a request for the *EquipmentNode* to provide the resolution of all *gid*s in a hierarchy. The *EquipmentNode* shall resolve the recipe hierarchy exactly the same as if the recipe was to be executed at the time of the *resolvePDE()* request.

8.4.2.11.2 The structure of a recipe can be determined by successively resolving the *ReferencedPDEs* at a level of the hierarchy and then moving downward to the next level. Note that the resolution of a *gid* at one level will affect the recipe structure at the next level down. The response to the *resolvePDE*() service shall comprehend the entire hierarchy below the *targetPDE*.

8.4.2.11.3 While the *targetPDE* will typically be a *Master PDE*, there is no requirement for this to be the case. This service can be used to resolve a subset of a complete recipe. The *targetPDE* shall contain a *gid* or a *uid* value.

8.4.2.11.4 The *inputMap* parameter specifies a list of *PDE* references with the corresponding *uid* that resolves each. This is a list of client specified resolutions for the *gid*s that may be encountered during the resolution process. If,

during this process, the *EquipmentNode* encounters any of the *gid*s listed in the *inputMap*, it shall use the corresponding *uid* from the *inputMap* in its resolution. This *inputMap* allows the client to override the normal resolution method by the equipment (see ¶8.5.1.3).

8.4.2.11.5 Some *gid*s listed in the *inputMap* may not be encountered during the resolution process. This is not considered an error and shall be ignored by the *EquipmentNode*. If a *gid* from the *inputMap* appears in the recipe hierarchy, but the corresponding *uid* is not available on the equipment, then the *EquipmentNode* shall note the situation as an error (see *resPDEstat*) and resolve that *gid* as if it had not appeared in the *inputMap*.

8.4.2.11.6 If *ResolvePDEreferences* is set to false (see ¶8.5.1.3), then only the entries in the *inputMap* are used to resolve the specified *PDE*. In this case, if a needed *gid* reference in the recipe structure is not included in the *inputMap*, it shall be treated the same as if no member of the group were available in the equipment store of *PDEs* (*resPDEstat* = MissingReferencedPDE)."

8.4.2.11.7 The *outputMap* parameter lists each *PDE* reference found during the resolution process and the resolution for each. The *targetPDE* is a *PDE* reference and shall be included in the *outputMap*. The *targetPDE* may contain a *gid* value.

8.4.2.11.8 If any *gid* from the recipe structure cannot be resolved to a *uid*, that *gid* shall appear in the *outputMap* with a zero length string for the corresponding *uid*. This error shall also be noted in *resPDEstat*.

8.4.2.11.9 All *outputMap* values that contain no errors shall be acceptable to the equipment as *inputMap* or *PDEmap* values.

Table 25 *resolvePDE*() Service Parameters

Parameter Name	*Req/Ind*	*Rsp/Cnf*	*Comment*
targetPDE	M	-	The uid or gid of the PDE to be resolved.
inputMap	M	-	Client specified resolutions for selected PDE references.
outputMap	-	M	Service provider generated list of all PDEs referenced in the recipe hierarchy beneath (and including) the targetPDE.
resPDEinfo		M	Report any problems resolving the PDE references.

8.4.2.12 *verifyPDE*()

8.4.2.12.1 The *verifyPDE*() service is used to check the validity of a specified *PDE*. It may be used to check a single *PDE* or all *PDEs* that it directly or indirectly references (see *verifyDepth*). Multiple levels of verification are available (see *verifyType*). The highest level, "Validity" is dependent on the supplier. The supplier shall document the verification steps it takes when *verifyType* is set to "Validity".

8.4.2.12.2 The verification process yields only the current status of a *PDE*. A successful verification does not guarantee that a later verification (or execution) will succeed.

8.4.2.12.3 Verification results shall be independent of the status of the equipment and its components. For example, if a processing module is out of service, it shall not change the outcome of the verification process. Equipment status is out of scope of this specification.

8.4.2.12.4 When *verifyDepth* is "All", the *verifyPDE*() service must resolve any *PDE* references that contain *gids*. The *inputMap* parameter allows the client to specify the resolution for some or all of these *gids.*

Table 26 *verifyPDE*() Service Parameters

Parameter Name	*Req/Ind*	*Rsp/Cnf*	*Comment*
targetPDE	M	-	The uid of the PDE to be verified.
inputMap	M	-	Client specified resolutions for selected PDE references.
verifyType	M	-	Which type of verification to be done.
verifyDepth	M	-	Should referenced PDEs also be verified?
verifySuccess	-	M	Overall success or failure.
verifyInfo	-	M	Lists results of the verification for each PDE.

8.4.2.13 *TransferContainer*

8.4.2.13.1 Some of the RaP services reference a *TransferContainer*. This section defines the *TransferContainer*.

8.4.2.13.2 The *TransferContainer* bundles one or more *PDEs* together for transfer. This provides messaging efficiency, logical grouping of *PDEs* where appropriate, and also creates a potential for data compression of the *PDEs*.

8.4.2.13.3 This specification allows any combination of *PDEs* to be included in the *TransferContainer*. They need not be related. However, the user may choose logical groupings. For instance, the FICS may choose to always download *Master PDEs* and all their related *PDEs* together.

8.4.2.13.4 The details of data compression are left as an implementation decision. For example, if the implementation chose to represent *PDEs* as files, it might be logical to implement the *TransferContainer* in the form of one of the popular file compression formats.

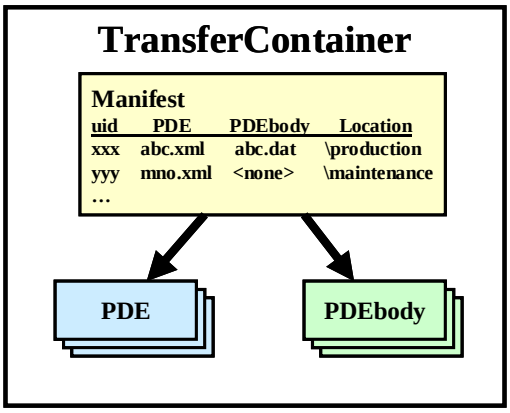

Figure 9
TransferContainer Illustration

8.4.2.13.5 Figure 9 provides an illustration of a *TransferContainer*. It contains a set of *PDEs*, a set of independent *PDEbodies* (as needed) and a *Manifest* that describes the contained *PDEs*.

8.4.2.13.6 The *PDEs* and the *PDEbodies* are in document form for transfer. The form of these documents is left as an implementation detail. It is assumed that the document form of the *PDE* will have some sort of descriptor (for example, a filename).

8.4.2.13.7 The *Manifest* associates the *uid* of each *PDE* contained with the descriptor of the document form of that *PDE* and the descriptor of its corresponding *PDEbody* (if not contained inside the *PDE*). It is arranged as a series of entries, each of which contains a single *uid*, the descriptor of the corresponding *PDE*, the descriptor of the *PDEbody* (if applicable), and a Location string. The Location string is not required, but it allows the client to specify where the recipe is to be stored (or the equipment to specify where it had been stored). The format of the Location string is supplier specified. Each of these entries in the *Manifest* allows the receiver of the Transfer Container to quickly identify the *PDE*, find its corresponding *PDEbody*, and store it appropriately. See Table 27 for more information on Manifest entry data.

Table 27 Manifest Entry Data Definitions

Manifest Data Item	*Description*	*Format*
uid	Value of the *uid* attribute of the *PDE* contained in the *TransferContainer.*	UUID
PDEdescriptor	Descriptor of the transfer form of the *PDE* (for example, a file name).	string
PDEbodyDescriptor	Descriptor of the transfer form of the *PDEbody* (for example, a file name).	string
Location	String that indicates the location on the equipment for this recipe. The format of this string is left to the implementer.	string

8.4.2.13.8 *Manifest* and the *TransferContainer* exist only for the purpose of transfer of *PDEs*. They are not tracked and cannot be accessed using any RaP defined interfaces outside of the context of the transfer operation for which they are created.

8.5 *RaPnode Requirements and Clarifications*

8.5.1 This section defines and clarifies requirements on the *RaPnode*s.

8.5.2 *General RaPnode Requirements And Clarifications*

8.5.2.1 A *RaPnode* is a provider of RaP defined services. Clients of RaP services are not required to be *RaPnode*s. However, in many cases *RaPnode*s will be clients of other *RaPnode*s.

8.5.2.2 Of the three defined *RaPnode*s, RaP requires only that an *EquipmentNode* be present and fully support the RaP services defined for it.

8.5.2.2.1 The other two types of *RaPnode*s may be present and if so, they shall fully support the RaP services defined for them.

8.5.2.3 *RaPnode*s may be combined into a single node. For example, an *EditorNode* may be combined with an *EquipmentNode*. This combined node shall support the superset of capabilities defined for these two types of nodes.

8.5.2.3.1 The combined *RaPnode* entity is a single node, therefore there is no requirement that RaP services be used internally (for instance, between the internal *EditorNode* and the internal *EquipmentNode* capabilities).

8.5.2.4 When a *PDEbody* is opaque to the user (for example, of a binary format), it shall be separated from the *PDE*/*PDEheader* construct as described in ¶8.3.7.

8.5.2.4.1 When the *PDEbody* is contained within the *PDE*, it shall meet the same format requirements as the *PDEheader.* In this case, its contents shall be comprehensible by the user.

8.5.2.4.2 The format requirements for the *PDE* are defined in subordinate specifications (sometimes called “dot standards”).

8.5.2.5 When a *TransferContainer* is received, the *RapNode* shall extract the *PDEs* and store them in such a way that each *PDE* can be exactly re-created for later transfer such that the calculated checksum will be correct.

8.5.2.6 RaP does not address and is not affected by internal design decisions for the Equipment, *PDEeditor*, and FICS. It deals only with communication among the Equipment, the *PDEeditor*, and the FICS.

8.5.2.6.1 For example, a *PDEeditor* might provide a “thin client” interface in the form of a web browser-based implementation. RaP does not restrict such designs. In this case, the *PDEeditor* should still provide a *RapNode* that complies with this specification.

8.5.2.7 For all checksum values defined by this specification, the MD5 method for computing checksums shall be used. MD5 is defined by the IETF (see the reference in §10).

8.5.3 *FICSnode Requirements And Clarifications*

8.5.3.1 The factory environment is not required to provide a *FICSnode* or the associated services.

8.5.3.1.1 RaP supports a communications structure where all communications between FICS and the Equipment and *PDEeditor* are initiated by the FICS.

8.5.3.1.2 Improved efficiencies may be possible when other *RaPnodes* can utilize *FICSnode* services. RaP supports, but does not require this approach.

8.5.3.2 If an *FICSnode* is provided, it shall support all the services defined for it as noted in Table 15.

8.5.4 *EquipmentNode Requirements And Clarifications*

8.5.4.1 The Equipment shall provide a single *EquipmentNode* that supports all the services defined for it as noted in Table 15.

8.5.4.2 The *EquipmentNode* shall be able to perform all RaP related activities without the presence of an *FICSnode* or an *EditorNode* (that is, where all communications with the *EquipmentNode* are initiated by the client).

8.5.4.2.1 The *EquipmentNode* should provide alternate methods of performing these activities that take advantage of the services provided by *FICSnodes* and/or *EditorNodes.*

8.5.4.3 The *EquipmentNode* shall supply values for all *PDEparameters* of the *Master PDE* as needed by soliciting these values as input parameters to the definition of processing jobs.

8.5.4.3.1 These input parameters shall have the same name as the corresponding *PDEparameters*.

8.5.4.4 The equipment supplier shall determine what parameters (that is, Module Parameters) to make available for setting by the *PDEs*.

8.5.4.4.1 The equipment supplier shall document all Module Parameters.

8.5.4.4.2 The equipment supplier is encouraged to make available all settings that effect the outcome of processing.

8.5.4.4.3 Some settings, such as calibration settings, may not be accessible for setting by the recipe.

8.5.4.4.4 All collections of settings or instructions (scripts, support files, etc.) associated with the processing specification shall be *PDEs*.

8.5.4.4.4.1 If this *PDE* contains settings not available for editing by the user (for example, calibration values), the equipment/*PDEeditor* may place appropriate restrictions on the *PDE*. For example, there might be a *PDE* (or *PDE* group) that contains calibration values that can be modified only by maintenance personnel and that cannot be downloaded to the equipment. In some cases, such a calibration *PDE* is automatically updated just before the run and can be specified only by *gid*.

8.5.4.5 The *EquipmentNode* shall be able to positively confirm the association of a *PDEheader* to its external *PDEbody* based on matching of information contained in the *PDEbody* with public information contained in the *PDEheader*.

8.5.4.5.1 The suggested method is to embed the *uid* inside the *PDEbody*, but this is not a required solution.

8.5.4.5.2 This may also be accomplished by computing the checksum of the *PDEbody* and comparing it to the *bodyChecksum* value contained in the *PDE*. This method requires no change to the *PDEbody*.

8.5.4.6 The *EquipmentNode* shall verify each *PDE* upon receipt. This verification shall be the equivalent of that initiated by the *verifyPDE*() service with *verifyType* = validity and *verifyDepth* = single.

8.5.4.6.1 The EquipmentNode shall reject any PDE that fails this verification process.

8.5.4.6.2 The equipment is encouraged to verify any recipes that are placed on the equipment from any source and via any method (for example, using removable media).

8.5.4.6.3 The equipment is encouraged to verify the Master *PDE* before execution. It is suggested that the verification performed be the equivalent of that initiated by the *verifyPDE*() service with *verifyType* = checksum and *verifyDepth* = all. Note that no results can be reported to the client. Any *PDE* that fails verification (except for the non-presence of a referenced *PDE*) should be removed from the recipe store on the equipment. If possible, a copy should be kept in a private area for debugging purposes.

8.5.4.7 The *EquipmentNode* shall monitor the collection of *PDEs* available at the equipment. When the collection changes (that is, a *PDE* is added or removed), then the *EquipmentNode* shall notify the FICS via the event data collection mechanism available on that equipment. Two events shall be defined to support this capability: *PDEadded* and *PDEremoved*. When multiple *PDEs* are added or deleted at the same time, the change shall be reported with a single event. A data collection parameter, *PDElist*, shall be provided to contain list of the *uid's* of the affected *PDEs*.

8.5.4.7.1 The *EquipmentNode* shall define a Status Variable that contains the time and date of the last change of the recipe collection on that equipment. This Status Variable shall be accessible by the FICS (for example, using the S1,F3/F4 message defined in SEMI E5).

8.5.4.7.2 In the event that a client is disconnected from the *EquipmentNode* and then reconnected, the client may check the above resynchronize by using the *getPDEdirectory()* service.

8.5.4.8 The *EquipmentNode* shall provide a configurable Boolean setting named "*ResolvePDEreferences*".

8.5.4.8.1 This setting shall be configurable by the FICS (for example, using the S2,F15/16 message defined in SEMI E5).

8.5.4.8.2 If *ResolvePDEreferences* is set to "True", the *EquipmentNode* shall resolve any *PDE* references left unresolved by the client when executing a recipe or processing the *resolvePDE*() and *verifyPDE*() services. When the *EquipmentNode* is required to resolve any unresolved *gid*'s, it shall select the available *PDE* with the same *gid* that has the newest *createDate*. In the rare instance that more than one *PDE* has the same (newest) creation date, the equipment shall select one whose *uid* value comes first alphanumerically.

8.5.4.8.3 If *ResolvePDEreferences* is set to "False", the *EquipmentNode* shall not resolve any *PDE* reference.

8.5.4.9 The *EquipmentNode* shall provide a predefined input parameter (Variable Parameter) for any job that executes a recipe consisting of *PDEs*. This Variable Parameter shall be named "PDEmap". The equipment client shall be allowed to supply this value during definition of the job. The format of the PDEmap shall be the same as the *inputMap* and *outputMap* parameters of the *resolvePDE()* service (see ¶8.4.2.3).

8.5.4.9.1 The PDEmap parameter value shall be used by the equipment to resolve the recipe structure to be used for the corresponding job. This process is the same as that used for the resolvePDE() service.

8.5.4.10 The *Equipment* shall provide at least one entity capable of creating *PDEs* (for example, *PDEeditor*), either a) on the equipment or b) directly communicating with the equipment

8.5.4.10.1 This *PDEeditor* shall be capable of adding or removing PDEs to the equipment's storage area. This may be done with RaP compliant services or by proprietary means. This requirement may also be met by having the PDEeditor and Equipment share storage space.

8.5.5 *PDEeditor and EditorNode Requirements And Clarifications*

8.5.5.1 A *PDEeditor* is any entity that can create *PDEs*. An *EditorNode* is a *PDEeditor* that supports RaP services. Not all *PDEeditors* are *EditorNodes*.

8.5.5.2 *PDEeditor*

8.5.5.2.1 This section discusses requirements and clarifications related to the creation of *PDEs*.

8.5.5.2.1.1 A *PDEeditor* shall be capable of creating and modifying all *PDEs* needed by the equipment.

8.5.5.2.1.2 For any recipe it creates, the *PDEeditor* shall ensure that any *ExecutionTarget* definition shall describe only equipment that are able to execute this *PDE*.

8.5.5.2.1.2.1 The *ExecutionTarget* attributes of *supplier*, *make*, *model*, and *recipeType* are used for this purpose.

8.5.5.2.1.3 All *PDEeditors* shall be able to positively confirm the association of a *PDEheader* to its external *PDEbody* based on matching of information contained in the *PDEbody* with public information contained in the *PDEheader*.

8.5.5.2.1.4 RaP does not restrict recipe creation to *EditorNodes* nor to only *RaPnodes*. Recipes can be created by any entity so long as the resulting *PDE*(s) meet all requirements of this specification.

8.5.5.2.1.5 During the *PDE* creation process, the *PDEeditor* shall provide the user the opportunity to enter *userInfo* text strings.

8.5.5.2.1.6 The *PDEeditor* shall enforce restrictions on *PDEs* in the same group. All *PDEs* in a group shall have the same set of *PDEparameters* and they shall have default values on the same subset of these *PDEparameters*. The *PDEeditor* might need access to a member of the group (or key information from a group member) in order to create a new member.

8.5.5.2.1.7 When the PDEeditor creates a new *PDE* the following shall be true with regard to *PDE* groups:

8.5.5.2.1.7.1 The PDEeditor that creates the *PDE* shall determine the gid during *PDE* creation

8.5.5.2.1.7.2 The PDEeditor shall determine (based on user input) whether a *PDE* will join an existing *PDE* group and if so, will enforce the rules for *PDEs* that are in the same group (see ¶8.3.2 for more information).

8.5.5.2.1.7.3 In all other cases, the *gid* shall be a new uuid value generated at recipe creation time (distinct from the *uid*).

8.5.5.2.1.8 During the *PDE* creation process, the *PDEeditor* shall allow the user the option of setting *ExecutionTargets* or omitting them. *ExecutionTargets* are intended to ensure that *PDEs* are executed on the correct equipment. By omitting the *ExecutionTargets*, the user chooses to bypass this protection by the equipment.

8.5.5.2.1.9 The *PDEeditor* shall allow the user to define the *PDEparameters* for a *PDE* and when, during processing, these *PDEparameters* shall be applied.

8.5.5.3 *EditorNode*

8.5.5.3.1 This section discusses requirements and clarifications related to communication with an *EditorNode*.

8.5.5.3.1.1 If *EditorNode*(s) are provided, each shall support all the services defined for it as noted in Table 15.

8.5.5.3.1.2 An *EditorNode* may choose to communicate with an *EquipmentNode* using RaP services, but it is not required to do so.

8.5.5.3.1.2.1 A proprietary interface may be used between *PDEeditor* and Equipment.

8.5.5.3.1.2.2 In either case, the equivalent of all RaP services defined between *EditorNode* and *EquipmentNode* shall be provided.

8.5.5.3.1.3 The *EditorNode* shall be able to perform all RaP related activities without the presence of a *FICSnode* (that is, where all communications with the FICS are initiated by the FICS).

8.5.5.3.1.3.1 The *EditorNode* should provide alternate methods of performing these activities that take advantage of the presence of a *FICSnode*. For example, at the *PDEeditor*, human triggered (*EditorNode* initiated) *PDE* upload may be supported.

8.5.5.3.1.4 The *EditorNode* can create a new *PDE* group at any time. There are no restrictions.

9 Compliance

9.1 Table 27 is intended to be used to document compliance to RaP. One copy of the table shall be completed per RaPnode to be implemented.

9.2 The first row of the table declares the type of RaP node to be delivered. The proper box shall be checked. Where two types of RaP node are combined, two boxes shall be checked. The combination of Equipment and FICS is not allowed.

9.3 Each (non-header) row in the table references a subsection number from §8 (RaP Requirements). A check in the checkbox for the "RaP Compliant" column shall indicate that all requirements in the named subsections for that row (and their subsections) are met.

9.4 For an equipment to be considered RaP compliant, it shall supply an *EquipmentNode* that complies with all applicable RaP requirements and in addition, all *EditorNodes* that are supplied (if any) shall be RaP compliant.

Table 28 RaP Compliance Table

RaPnode Type: EquipmentNode ☐ EditorNode ☐ FICSnode ☐		
Section	*Topic*	*RaP Compliant*
PDE Content		
8.3	*PDE* Class Diagram	☐
8.3.1.1	*PDE* Class	☐
8.3.2	*PDEheader* Class	☐
8.3.3	*AntecedentData* Class	☐
8.3.4	*ReferencedPDE* Class	☐
8.3.5	*ExecutionTarget* Class	☐
8.3.6	*PDEparameter* Class	☐
8.3.7	*PDEbody* Class	☐
8.3.8	*PDEbodyReference* Class	☐
RaPnode Requirements		
8.5.2	General *RaPnode* Requirements	☐
8.5.3	*FICSnode* Requirements	☐
8.5.4	*EquipmentNode* Requirements	☐
8.5.5	*EditorNode* Requirements	☐
RaP Services Requirements		
8.4.2.1	*RaPnodes*	☐
8.4.2.2	*Required Services Per Node*	Compliance represented in service compliance below.
8.4.2.3	*Service Message Parameters*	Compliance noted below for each service using these parameters.
8.4.2.5	*getPDEdirectory*()	☐
8.4.2.6	*deletePDE*()	☐
8.4.2.7	*getPDEheader*()	☐
8.4.2.8	*getPDE*()	☐
8.4.2.9	*requestToSendPDE*()	☐
8.4.2.10	*sendPDE*()	☐
8.4.2.11	*resolvePDE*()	☐
8.4.2.12	*verifyPDE()*	☐
8.4.2.13	*TransferContainer*	☐

10 Related Documents

Grady Booch, James Rumbaugh, and Ivar Jacobson, *The Unified Modeling Language User Guide*, Reading Massachusetts, Addison Wesley: 1999 ISBN 0-201-57168-4.

Rivest, Ronald, *The MD5 Message-Digest Algorithm*, IETF RFC: 1992 http://www.ietf.org/rfc/rfc1321.txt.

Uniform Resource Name (URN) Syntax, IETF RFC 2141, May 1997 http://www.ietf.org/rfc/rfc2141.txt.

RELATED INFORMATION 1

NOTICE: This related information is not an official part of SEMI E139 and was derived from North American Information and Control Committee. This related information was approved for publication by full letter ballot on December 10, 2004.

R1-1 Scenarios

R1-1.1 This section contains example scenarios to illustrate how RaP services might be used in practice. They represent only a subset of possible scenarios. The purpose of including these scenarios is to demonstrate how the RaP services can work together to perform complex tasks.

R1-1.1.1 For these scenarios, certain assumptions were needed about how business rules of the FICS and about the outcome of certain services. These assumptions are stated. The FICS business rules are not intended to represent actual factory systems. Instead, they are designed to demonstrate possible combinations of services to perform a task. RaP is intended to work with a wide range of FICS business rules. It is not possible to demonstrate all situations.

R1-1.1.2 Assumptions for all scenarios: (unless otherwise noted in the individual scenario)

- Equipment, Editor, and FICS nodes are separate nodes that communicate via RaP services (note that Equipment to Editor communications are not required to use RaP services).
- The "User" entity represents any source of stimulus, human or electronic.
- All message receive positive or "success" (response unless otherwise noted).
- The provided scenarios may not be the only way to accomplish each task. This set is not meant to be exhaustive.
- Response messages are not shown unless they add information for the reader.

R1-2 Scenario – Download PDEs

R1-2.1 This scenario is representative of most uses of the *sendPDE()* service.

R1-2.1.1 Assumptions:

- The *PDEs* to be downloaded are not currently stored on the equipment.

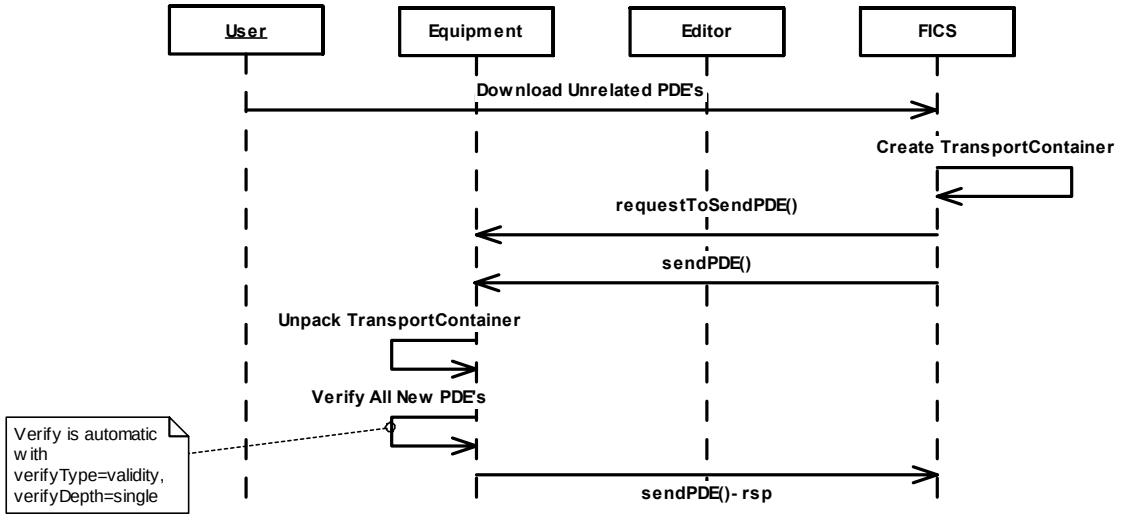

Figure R1-1

R1-2.1.1.2 Notes:

- The *EquipmentNode* is the only one required to validate upon receipt of a TransportContainer. Transfer to FICS or to Editor would be similar, but with the internal Verify step left off.
- Whether the *PDEs* are related (e.g. same recipe) or not does not affect this scenario. If all *PDEs* for a recipe are available, a subsequent scenario to validate the full recipe can be executed.

R1-3 Scenario – Upload Selected PDEs

R1-3.1 This scenario is representative of most uses of the *getPDE()* service.

R1-3.1.1 Assumptions:

- The equipment stores some *PDEs*. It may or may not be the "System of Record".
- The user is searching for particular *PDEs*, based on some unspecified criteria that is beyond what can be returned in *getPDEdirectory()*.
- Once the *PDEs* are identified, they are to be uploaded to the FICS.

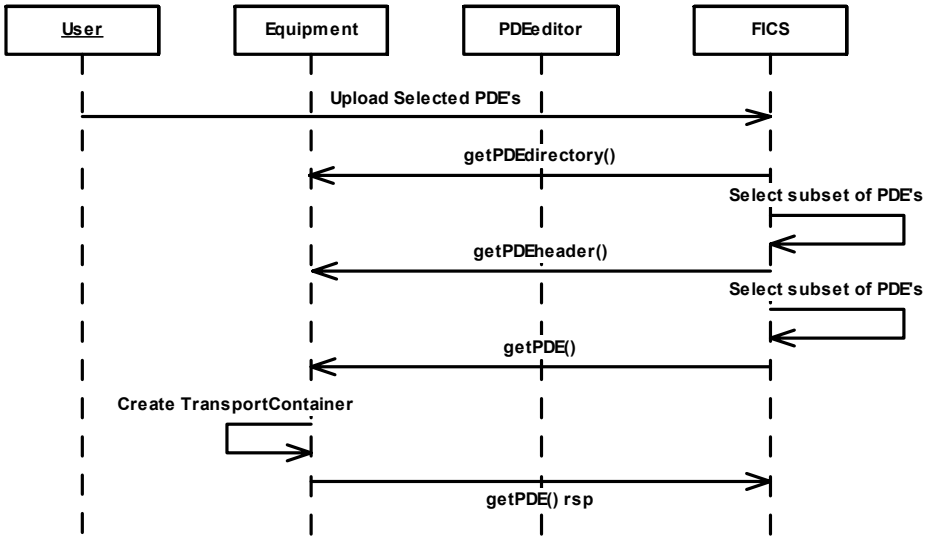

Figure R1-2

R1-3.1.1.2 Notes:

- The *getPDEdirectory()* service is used to get a complete list of *PDEs* available.
- The *getPDEheader()* service is used to get detailed information about a subset of the *PDEs* available.
- The final *PDEs* uploaded are selected based on the *PDEheader* information.

R1-4 Scenario – Delete Selected PDEs

R1-4.1 This is a common usage of the *deletePDE()* service, but does not assume pre-knowledge of which *PDEs* are actually on the equipment.

R1-4.1.1 Assumptions:

- The user knows which *PDEs* should be on the equipment. So he/she needs to see a list of *PDEs* actually on the equipment and will delete those that are not needed.

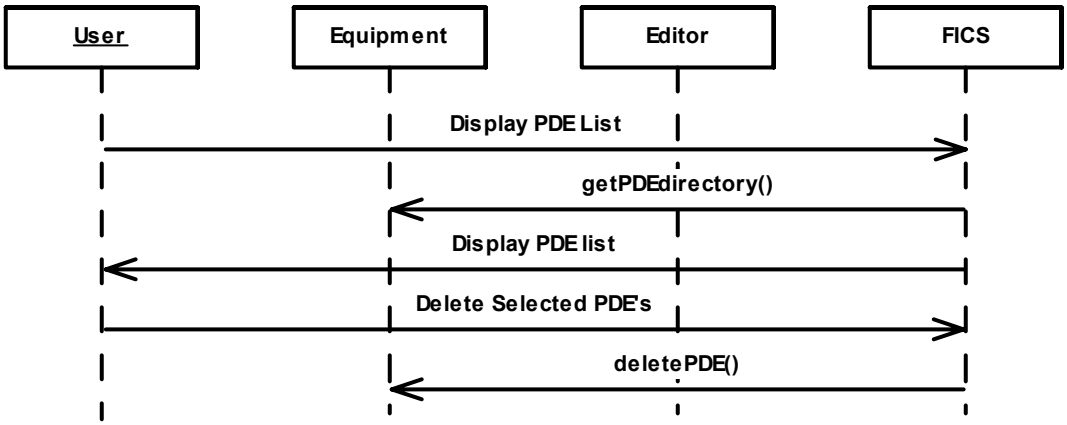

Figure R1-3

R1-4.1.1.2 Notes:

- The *getPDEdirectory*() service can return basic information about the *PDEs*, including *name*, *uid*, *description*, *createDate*, etc. as needed to support the identification of *PDEs*.
- Systems that track which recipes are on the equipment may not need to request a directory listing. Such tracking is possible by monitoring the event reports when the recipe collection changes and checking the status variable that notes the last time the recipe collection changed.

R1-5 Scenario – Create PDE

R1-5.1 There are many different ways to approach this scenario. This is just one of those ways. It is intended to demonstrate most of the services that might be used.

R1-5.1.1 Assumptions:

- The equipment is the "system of record" for *PDEs* in this example.
- This *PDE* is created based on an existing *PDE* that is stored on the equipment.

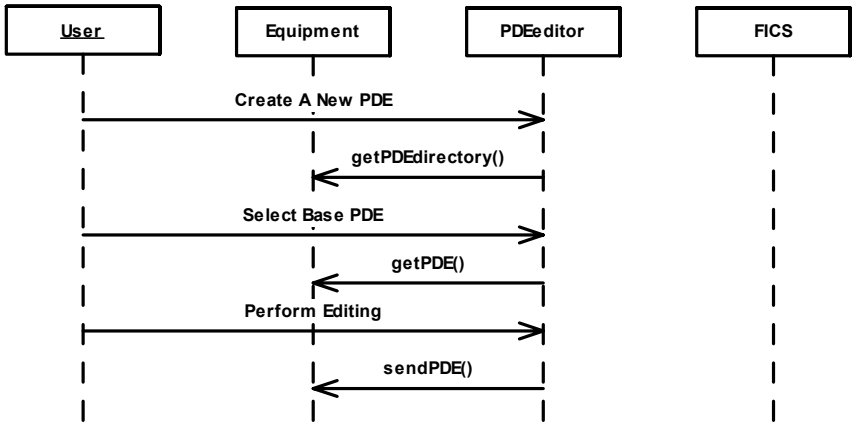

Figure R1-4

R1-5.1.1.2 Notes:

- If the existing & resulting *PDEs* were stored by the FICS, the scenario would be similar (replace Equipment with FICS).

- "Editing" a *PDE* is the same as creating a new *PDE*, since each new version must be given a new unique identifier.
- The *PDEeditor* is responsible for verification of the new *PDE*. Resolution of verification issues is between *PDEeditor* and User.

R1-6 Scenario – Preparation for Processing (1)

R1-6.1 This scenario represents a central recipe management system with off-tool system of record.

R1-6.1.1 Assumptions:

- The equipment is configured to never resolve *gid* references in recipes. The host must/will supply all needed resolutions.
- The FICS (per its Recipe Management System) knows exactly which *PDEs* should be used in any processing situation.
- All *PDEs* used the last time that this process was run are on the equipment.
- The user has updated one *PDE* that must replace an older version; this *PDE* and its uses are managed by an FICS-level recipe management system.
- The automatic verification of the downloaded *PDE* is assumed to succeed.
- While RaP recommends that an equipment perform a verification of the full recipe before executing it, this equipment does not perform that task.
- A Process Job will soon use this newly verified *Master PDE.*

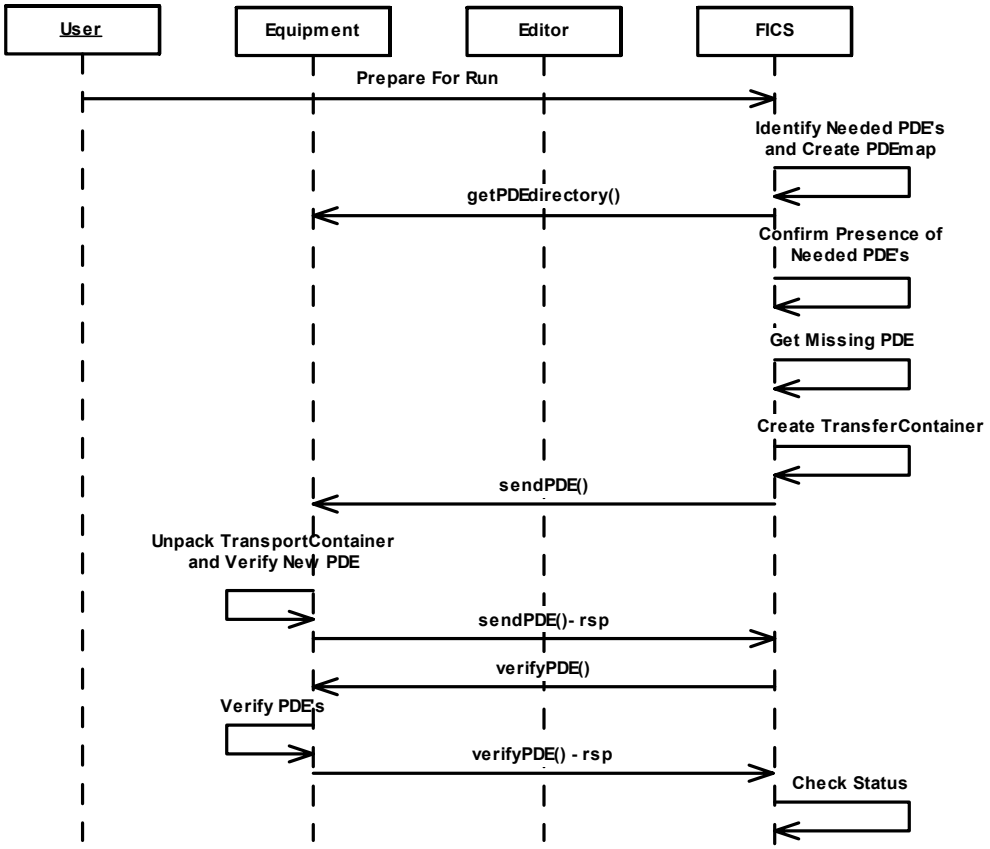

Figure R1-5

R1-6.1.1.2 Notes:

- The *getPDEdirectory()* service is used to determine what *PDEs* are on the equipment.
- The FICS accesses the recipe management system to obtain the list of "needed *PDEs*" This is compared with the *PDE* directory to determine which, if any, are needed.
- The *TransferContainer* contains only the *PDE* not currently on the equipment.
- The *verifyPDE()* service refers to the *Master PDE* for the intended process job and has *verifyType* = Checksum and *verifyDepth* = All. A full *inputMap* is supplied.
- The *PDEmap* is used as the *inputMap* for the *verifyPDE()* service.
- In this scenario, the "*requestToSendPDE()*" message is omitted. Its use is at the option of the client to ensure a *PDE* download will not be rejected.

R1-7 Scenario – Preparation for Processing (2)

R1-7.1 This scenario is intended to represent some aspects of early stage recipe management where recipes are stored on the equipment and central control and tracking is not available.

R1-7.1.1 Assumptions:

- *PDE*s are created at the equipment, which serves as the system of record.
- The business practice of the FICS is that the latest version of each *PDE* available on the equipment should always be used. For that reason, *PDEs* are referenced by *gid* and the equipment always resolves these *gid*s.
- The FICS records which *PDEs* were last used with each *Master PDE*. It detects when the current set differs from the previous execution of this *Master PDE* and seeks confirmation from the user that the change is appropriate.
- All *PDEs* used the last time that this *Master PDE* was executed are on the equipment.
- The user has updated one *PDE* that must now be used in place of an older version.
- The equipment follows the RaP recommendation that an equipment perform a verification of the full recipe before executing. Therefore, such verification is not necessary in this scenario.
- A Process Job will soon use this newly verified *Master PDE*.

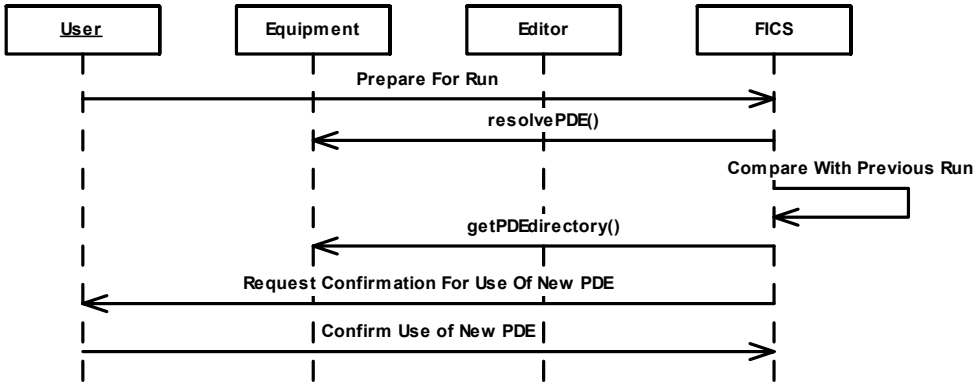

Figure R1-6

R1-7.1.1.2 Notes:

- The *Master PDE* is specified as the *targetPDE* in the *resolvePDE()* message.

- The *getPDEdirectory()* service is used to get further information about the new *PDE* found in the *outputMap* of the *resolvePDE()* service. Alternately, the *getPDEheader()* service could have been used if more detailed information were required.

RELATED INFORMATION 2

NOTICE: This related information is not an official part of SEMI E139 and was derived from the North American Information and Control Committee. This related information was approved for publication by full letter ballot on December 10, 2004.

R2-1 Implementation Suggestions

R2-1.1 This related information is included to supplement the specification with some suggestions about the use of RaP and its capabilities. Its purpose is a) to promote more uniform application of RaP and b) to share some insights of the creators of RaP that may lead to improved results from its use.

R2-2 Combining RaPnodes

R2-2.1 ¶8.5.2.3 mentions the possibility of combining *RaPnode*s. This section will briefly discuss this topic to outline when and why this might be appropriate.

R2-2.1.1 In general, when multiple nodes are combined, efficiency can be gained by:

- Coding the individual services only once (since most are common to all *RaPnode*s)[6].
- Sharing the computing platform where the software is run.
- Eliminating the communication between the two separate *RaPnode*s by designing them to be a single entity.

R2-2.1.1.1 Not all *RapNode* combinations are practical. In particular, the combination of *EquipmentNode* with *FICSnode* is not commercially feasible due to the many different suppliers' equipment that must communicate with one FICS. The other two combinations may be useful in the some situations.

R2-2.1.1.2 Combining the *EquipmentNode* and the *EditorNode* mirrors the traditional equipment "on-tool" editing capability. This approach might provide the easiest transition to RaP for some equipment suppliers. It also ensures that the equipment will function "out of the box" without integration. Since multiple *PDEeditors* are allowed, the "off-tool" version could be provided later or as an add-on.

R2-2.1.1.3 The combination of *FICSnode* with *EditorNode* might represent the integration of an off-tool editor with an off-tool recipe management capability. In the long term, there might be a movement to general *PDEeditors* that are compatible with many suppliers' equipment recipes. Integrating a recipe management capability with that general editor would be a logical next step.

R2-2.1.1.4 If the implementation for messaging is SECS-II only, then any equipment with an on-tool editor must combine the *EquipmentNode* and *EditorNode*. This is necessary since multiple SECS-II sessions are not well supported.

R2-2.1.1.5 When two *RaPnodes* are combined, they are expected to act as a single entity with a union of the capabilities of the two types of *RaPnodes*. The combined *RaPnodes* are expected to share resources, including the recipe storage area. Messages to the combined *RaPnodes* will not differentiate which node type is being addressed.

R2-3 RaP Support For SEMI E40 and SEMI E30

R2-3.1 RaP can be used in conjunction with SEMI E40 Process Job Management or with SEMI E30 GEM. This section discusses how to map the key recipe-related parameters from those two standards to the RaP attributes. In both cases, the management of the recipes uses RaP services. This section discusses how the recipe is specified for the processing activity and how the Variable Parameters are communicated.

R2-3.2 In SEMI E40, recipe information is specified as part of the job specification in the Stream 16 messages as RCPSPEC (recipe identifier - text) and RCPPARNM (recipe variable name - text). To satisfy RCPSPEC, a RaP implementation should supply the identifier (*uid* or *gid*) of the *Master PDE*.

R2-3.3 The Variable Parameters defined in RaP were intended to correspond directly with the RecipeVariables[7] defined in SEMI E40. The *PDEparameters* of the *Master PDE* should be mapped directly to RecipeVariables. The

6 Note that even if the *RaPnode*s are not combined, a single body of code could be created, with internal switches to enable the proper services for the particular type of *RapNode*.

RaP *PDEparameter* class includes a *name* attribute. This *name* value can be supplied in the Steam 16 messages as the value of an RCPPARNM parameter.

R2-3.4 Support for SEMI E30 works differently. In this case, there is no concept of a process job. Instead, it is a two step process: 1) the recipe is "selected" for execution, and then 2) a START command is given to begin execution. Both of these requests are made using the S2,F49 message (Enhanced Remote Command – see SEMI E5). The Remote Command message accepts parameter values (called CEPVAL's). Each CEPVAL has a corresponding identifier called CPNAME.

R2-3.5 In the first step, the RCMD (remote command) is RCP-SELECT. The identifier of the recipe is contained in a CEPVAL. The *uid* (or *gid*) of the *Master PDE* is supplied as the CEPVAL to identify the recipe. The supplier determines the CPNAME of this parameter.

R2-3.6 In the second step, the S2,F49 message requests the START remote command. In this message, the CEPVAL's contain parameters that apply to the activity. The equipment may have some pre-defined parameters. The *PDEparameter* values may also be specified here as CEPVAL's. The *name* of the *PDEparameter* should be specified as the CPNAME.

R2-4 Factory Assigned Recipe Attributes

R2-4.1 Some factory systems track various attributes of specifications within their factory. Recipes are one type of specification and often have important attributes associated with them. For example, a recipe may need to undergo certain tests before it is qualified to use in production. Also, there may be some sort of management sign-off process. These and other such attributes are used by the factory to help determine what recipe can be run in a given situation.

R2-4.2 The definition of recipe management practices of the factory is beyond the scope of RaP. However, a place has been provided within the *PDEheader* for the factory to store (and later retrieve) such information. The *userInfo* attribute provides a list of text strings that the factory can use for this purpose.

R2-4.3 Any attribute that can be represented as a text string can be included. Using common text formatting approaches (for example, comma separated values), both simple and complex attributes can be stored. For example, one *userInfo* field might contain a name/value pair something like this: "Approved, True".

R2-4.4 It is expected that the *userInfo* will not be used for very large data structures, such as photographic images. An attempt to include data much larger than the original *PDE* might cause unexpected results.

R2-4.5 Since the equipment will not try to interpret these values and undocumented changes to the recipe are not possible, this provides a secure place to store these values.

R2-4.5.1 Some examples of *userInfo* strings include:

- Comment — The user can enter a detailed explanation of the need for this *PDE*, the conditions that led to its creation, etc. This can help when transfer of duties to a new engineer occurs.
- Approval Level — is the recipe approved for use?; what use is it approved for?
- Version — what version number will the host assign to this *PDE* for tracking purposes?
- Classification — there are many ways to classify recipe components – the factory area of use, which step in the process, which process (or process version), etc.

R2-5 Supplier Assigned Recipe Attributes

R2-5.1 The equipment supplier may need to provide information about each *PDE* that is not standardized. The *supplierInfo* attribute is provided to allow the supplier to include this information. The format is the same as for *userInfo* – a list of text strings.

---

7 SEMI E40 also uses the term "Recipe Variable Parameters" to refer to RecipeVariables.

R2-5.1.1 Here are some examples of potential uses of *supplierInfo*:

- *formatVersion* — Supplier assigned string that documents the version of the format or language used to create this PDE. This can help the user determine whether the PDE is compatible with a particular equipment.
- *version* — Version of the *PDE*. This represents the supplier-assigned version value. If equipment-based recipe management is used, a versioning scheme may be helpful. The equipment supplier should document its versioning strategy.

R2-6 Parameterization Notes

R2-6.1 Parameterization of equipment recipes begins with the setting of Module Parameters according to Variable Parameters entered at run time. There are other uses of parameterization that may be very beneficial. Parameterization can also help with complex control situations. This section discusses such opportunities.

R2-6.1.1 *Selection of ExecutionTarget* — The recipe documents which *ExecutionTargets* are allowed to execute each *PDE*. At execution time, it is possible that more than one specified *ExecutionTarget* will be available (for example, identical processing modules in a cluster tool). The selection is made by the equipment, in accordance with any instructions in the recipe. However, there may be times when one of the processing modules is not operating within specifications for certain products. In such a situation, how can the factory force the equipment to exclusively choose the other process module? One answer would be to provide an input parameter to the recipe for selection of process module for this *PDE*. The default might be "use both" with the opportunity to select one or the other exclusively.

R2-6.1.2 *Process Control For Multiple ExecutionTargets* — There are situations where multiple, equivalent processing modules are available on the equipment and are all identified as *ExecutionTargets*. If wafers are sent to the next available process chamber, it would not be possible to predict which process module a given wafer might use. When process control is being applied to this equipment, this can cause complication. Each of the process modules can have very different control responses. So each would have different control settings for a specific material/job situation. To serve the user's needs in this case, it should be possible to define one set of *PDEparameters* for each possible *ExecutionTarget*. These parallel parameter sets would all need to be set. The *PDE* would select the proper parameter set based on the choice of process module for a given wafer.

R2-6.2 The reverse issue may also exist in this situation of multiple equivalent *ExecutionTargets*. Some ModuleParameters should take on the same value, no matter which *ExecutionTarget* is chosen. The implementation should allow for the specification of a single *PDEparameter* whose value will be applied correctly for any of the specified *ExecutionTargets*.

R2-6.2.1 So, the user should have the option to specify different values per *ExecutionTarget* or a single value to be used for all.

R2-7 Traceability

R2-7.1 Proper traceability of processing should include recording enough information to determine exactly what was done in any instance of processing. An important part of this information set is a record of the instructions provided to the equipment. This includes the specific *PDEs* used and the external setting values provided to satisfy the Variable Parameters.

R2-7.1.1 It is recommended that the equipment supplier create data collection parameters that allow the user to collect the following:

- The identifier of the *Master PDE* specified for the process job,
- The *PDEmap* provided to resolve the recipe structure,
- A list of all the *PDEs* actually used during processing,
- All Variable Parameter values supplied for the process job, and
- The *PDE*(s) used for each instance of processing (for example, when a wafer is processed in a process module.
    - Also include the *PDEparameter* settings that affected that instance of processing.

- The settings for all Module Parameters of this process module.

R2-8 Recipe Security

R2-8.1 The use of the universally unique identifier and checksum provide significant protection against misprocessing due to the use of the wrong recipe. However it is not completely secure from intentional tampering.

R2-8.2 It is possible for someone to modify an existing *PDE* without changing the *uid*. To complete the deception, a new checksum would need to be calculated and included in the *PDE*. A *PDE* changed in this way might not be detected in some factory systems.

R2-8.3 There is a way to address this problem and ensure that only the correct recipes are used. This assumes that there is a factory level recipe management system where the known-good *PDEs* are all logged and their authorization for use is recorded. This system should log the *uid* of each *PDE*, along with its *checksum*.

R2-8.4 Recall that the equipment will check each *PDE* against its internal *checksum* value when the *PDE* is transferred to the equipment and most will check it again before execution. This ensures that each *PDE* is self-consistent.

R2-8.5 Whenever a recipe is to be used for processing, the FICS can perform a *getPDEdirectory()* service to list the *PDEs* to be used and their *checksums*. It can then compare the stored (known-good) checksum for that *PDE* with the current *checksum* value. In this way, the FICS can detect any changes and ensure that the correct *PDEs* are being used.

R2-9 PDEeditor/Equipment Synchronization

R2-9.1 RaP defines services in support of an off-tool recipe editor. When the editor is a separate entity, there arises a potential problem of synchronization. Any modifications to the equipment software may require a corresponding change to the *PDEeditor*. If these updates are not done in a coordinated way, then significant problem could develop. RaP does not offer a solution to this problem. It provides only this warning of the potential problem. It is the responsibility of the users to put a process in place to ensure that their suppliers maintain synchronization of the equipment and editor software.

R2-10 Human Involvement In Recipe Selection

R2-10.1 RaP is designed to work well in an automated factory system. In some situations, however, humans must be involved in the process. Unfortunately, the UUID format is not user-friendly.

R2-10.1.1 When a system must rely on a human to select a recipe, there must be enough human-recognizable information to ensure the correct choice is made. This is a recommendation of how to support such a human selection of *PDEs*.

- *uid/gid* — It is not recommended that the *uid* and *gid* be displayed to the user. They are long strings with no human-parsable information.
- *name* — *Name* should be the most useful information to the user. However, it depends on the users defining a meaningful naming convention for *PDEs*. This is an important consideration.
- *groupName* — This can give the user a good idea of the intended use of this *PDE*. Again, it depends on a meaningful naming convention from the users.
- *type* — This allows the user to distinguish, for example, between a sequence recipe and a module recipe or between etch and clean recipes. The use of type may vary, but it will typically help make important distinctions.
- *createDate* — Allows a user to get a sense of the chronology.
- *description* — If the users enter appropriate information, this can be very helpful. However, it can also be very long. This is good for an extended display about a *PDE*.
- *author* — Who created a *PDE* is a very strong clue about what it is to be used for.
- *userInfo* — More good information for an extended display.

- *supplierInfo* — Again, good information for an extended display.

R2-10.2 In general, a display of the *name, groupName, type* and *createDate* should be adequate to support a recipe selection in most cases. An extended display should be available when more help is needed.

R2-11 Optimizing getPDEdirectory() Service

R2-11.1 A poor implementation of the *getPDEdirectory()* service may be inefficient. It is clear that some *getPDEdirectory()* requests may require the *RaPnode* to return information from the header of all the recipes stored at this node. Reading bits of information from hundreds of potentially large files can be time consuming. It is suggested that steps be taken to optimize this implementation. For instance, if key parts of the header from each *PDE* on the *RaPnode* are stored separately (for example, in a database), then retrieval of this information can be very fast. However, this cataloging of all recipes requires careful management of the *PDEs* and some overhead to maintain. The software designer should consider all these factors when implementing this service.

R2-12 PDEeditor Identification

R2-12.1 Not all *PDEeditors* are *EditorNodes*. There may be standalone editors that create *PDEs* that are moved to the equipment (or to the Recipe Management System) by other means – email, floppy disk, or other transport mediums. Some recipes may originate from the supplier. It is important that all *PDEeditors* embed meaningful nodeID's that identify the source clearly to the eventual user of the recipe. It is recommended that all nodeID's contain URN strings as defined in Internet standards.

SEMI E139.1-0705
XML SCHEMA FOR THE RaP PDE

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

Table of Contents

List of Figures

List of Tables

1 Purpose

1.1 The purpose of this document is to define an XML-based format for transferring recipe components. This includes an XML schema that corresponds to the UML model for the RaP Process Definition Element (PDE) as defined by SEMI E139. In addition, an XML schema is defined for the Manifest that must be included in the TransferContainer with the PDE's during transfer.

2 Scope

2.1 The scope of this document is the faithful representation of the PDE and Manifest definition from SEMI E139 in an XML schema. It will not add new domain information or concepts to the model. The only additions made are those needed to render a useful XML schema.

3 Limitations

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

4 Referenced Standards and Documents

4.1 *SEMI Standards*

SEMI E121 — Guide for Style & Usage of XML for Semiconductor Manufacturing Applications

4.2 *OMG Standards*

Unified Modeling Language (UML) Specification, Version 1.4, OMG Specification 01-09-67, (http://www.omg.org/technology/documents/modeling_spec_catalog.htm).

4.3 *W3C Standards*

Canonical XML Version 1.0 — W3C, 15 March 2001 (http://www.w3.org/TR/xml-c14n).

Extensible Markup Language (XML) 1.0 (Second Edition) — W3C, 6 October 2000 (http://www.w3.org/TR/2000/REC-xml-20001006/).

Namespaces in XML — W3C, 14 January 1999 (http://www.w3.org/TR/1999/REC-xml-names-19990114/).

XML Schema Part 0: Primer — W3C, 2 May 2001 (http://www.w3.org/TR/xmlschema-0/).

XML Schema Part 1: Structures — W3C, 2 May 2001 (http://www.w3.org/TR/xmlschema-1/).

XML Schema Part 2: Datatypes — W3C, 2 May 2001 (http://www.w3.org/TR/xmlschema-2/).

XML Path Language (Xpath) — W3C, 16 November 1999 (http://www.w3.org/TR/xpath/).

5 Terminology

5.1 *Abbreviations and Acronyms*

5.1.1 *UML* — Unified Modeling Language

5.1.2 *W3C* — World Wide Web Consortium

5.1.3 *XML* — eXtensible Markup Language

5.2 *Definitions And Acronyms*

5.2.1 *UML (Unified Modeling Language)* — A notation for representing object-oriented designs and views created by Booch, Rumbaugh, and Jacobson in order to merge their three popular notations plus aspects of other existing notations into a single object-oriented notation intended to be usable by all.

5.2.2 *XML (eXtensible Markup Language)* — A markup language used for representing data rich with context and content in documents and in communications. XML is an extension of SGML, a document-oriented markup language. It was created by W3C for use on the Internet. XML can represent object-oriented structures.

6 Conventions

6.1 This section discusses the conventions used in this specification for translating UML to XML and for documenting the XML. These conventions are heavily influenced by the SEMI E121 Guide for Style & Usage of XML for Semiconductor Manufacturing Applications.

6.2 The reader is expected to have a working knowledge of the UML, XML, and Schema specifications (See ¶4.2 and ¶4.3). This document does not provide tutorial information on these subjects.

6.3 *Translating UML to XML*

6.3.1 This document follows the guidelines for XML as outlined in SEMI E121 "Guide For Style & Usage of XML for Semiconductor Manufacturing Applications".

6.3.2 Some of the key guidelines in this specification are summarized here:

- Attributes of a class are generally represented as elements such that they can be easily extended by other applications.
- Inheritance is modeled as element extension.
- Compositions always turn into contained elements (or contained collections if multiples are allowed).
- Associations and Aggregations are modeled as contained elements or arrays wherever possible to simplify the instance model.

6.3.3 The translation of a UML class to XML is documented using a table format illustrated by Table 1. Note that the PDE model in SEMI E139 does not include services, so only attribute and association representations are addressed here. Any superclass information is addressed in the text and the diagrams.

6.4 *Translation Table Column Header Description*

- **Attribute or Role Name** — If an attribute, the name of the attribute is placed here. If an association (including aggregation or composition), the role name from the UML diagram is placed here. Compositions are often not assigned role names. In that case "none" is placed here.
- **UML Type** — If an attribute, the data type of the UML attribute is placed here. If an association, the type of association is placed here. The possible types are "Composition", "Aggregation", or the basic "Association". UML defines these three types of association.
- **XML Element or Attribute** — Lists the type of XML construct used to represent the UML attribute or association.
- **XML Name/Type** — Provides the name and data type of the resulting XML construct. The type may be a built-in type (for example, xs:string), or a named type defined within the XML schema.

Table 1 Example Translation Table

Attribute or Role Name	*UML Type*	*XML Element or Attribute*	*XML Name/Type*
friend	association	element	Friend: HumanReferenceArray
employees	aggregation	element	Employees: HumanArray
family	composition	element	Family: HumanArray
name	string	element	Name: xs:string

6.5 *Documenting The XML With Diagrams*

6.5.1 This document provides graphical representations of the included XML. Although no standard graphical notation for XML could be found, various XML tools have their own notation. This document will use the notation provided by XMLSPY® from Altova Corporation[1]. There is no requirement for use of XMLSPY® in order to

1 All images/graphics were created using Altova's XMLSPY®. Copyright 2003 Altova GmbH and reprinted with permission of Altova.

comply with this specification. Figure 1 shows a sample XML diagram that will be used to provide a basis for explanation of the XML graphical notation used in the rest of the document.

6.5.2 In the diagram, rectangular boxes represent XML elements. Ownership or containment is read from right to left in the diagrams. In the sample diagram, ParentType contains Child1, Child2, Child 3, and Child 4. In turn, Child2 contains Child2a, Child2b, and Child2c. The additional symbols (8-sided boxes) represent sequences or choices. See Table 2 for an explanation of these symbols.

6.5.3 Please note that "element", "sequence", "all", and "choice" are XML terms (see the W3C standards referenced in ¶4.3).

Table 2 Altova XMLSPY® Schema Diagram Symbols

	Denotes a required ordered sequence of the right hand elements with a cardinality of one for each element. (sequence)
	Denotes an optional ordered sequence of the right hand elements with a cardinality of one for each element. (sequence)
	Denotes a required, but unordered, sequence of the right hand elements with a cardinality of one for each element. (all)
	Denotes a required choice of the right hand elements. Exactly one or two of the right hand elements must be present. (choice)
	Denotes an element. Optional element if outline is dashed line.
	Denotes an element that contains parsed character data.

6.5.4 A graphic using a solid line is a required element; using a dashed line represents an optional element. Numbers or ranges in the lower right hand corner represent cardinality. The default cardinality is one.

6.5.5 To simplify a diagram or help focus on a particular aspect, detail may be hidden. The 8-sided symbols have a small square on the right end. If a minus sign "-" is in the box, then all detail is shown. If the box contains a plus sign "+", then all detail to the right of that symbol is hidden. The example has no hidden detail.

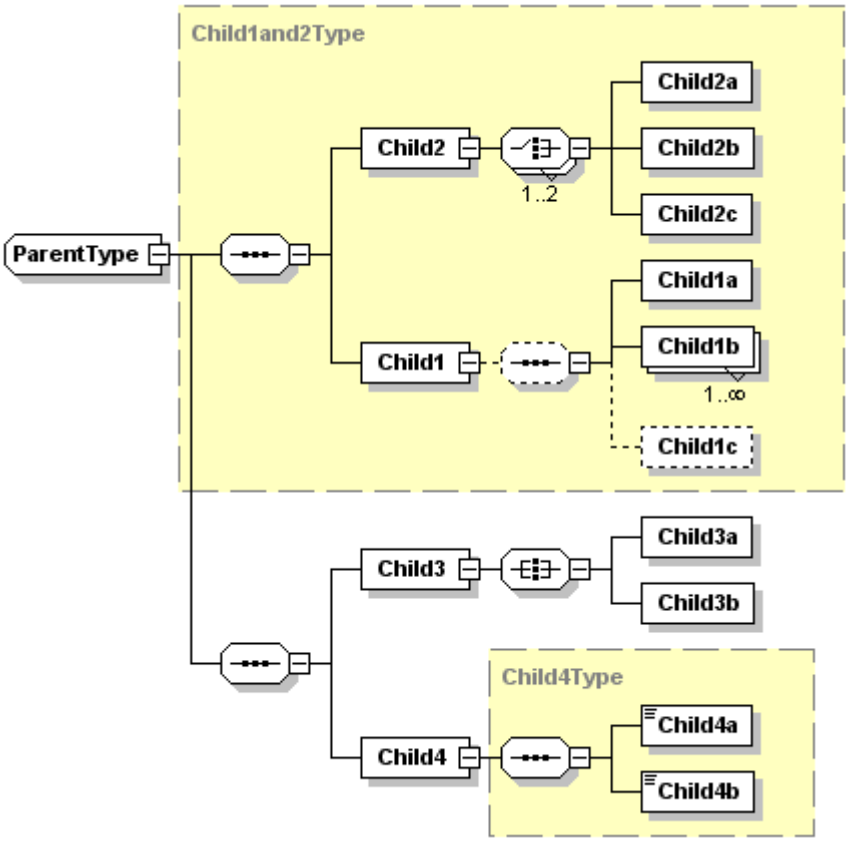

Figure 1
XML Example Diagram

6.5.6 The yellow (or gray if printed in monochrome) boxes indicate the use of other defined types. So, Child4 is of type "Child4Type". Child4Type defines Child4a and Child4b. This detail may be hidden in the diagram. Object oriented inheritance is typically represented in XML as type extension. In Figure 1, ParentType extends Child1and2Type by adding a sequence that includes Child3 and Child4.

6.5.6.1 Reading Figure 1 would yield the following additional information:

- Child1and2Type is an ordered sequence of two items: Child2 and Child1.
- Child1 contains an optional ordered sequence of Child1a, one or more Child1b, and (optionally) Child1c.
- Child2 contains a choice of one or two of the following: Child2a, Child2b, and Child2c.
- Child3 contains an unordered sequence of Child3a and Child3b.

6.5.7 *XML Schema Sample*

6.5.7.1 The sample XML for the example shown in Figure 1, is presented below. Refer to the XML documentation referenced in ¶4.3 for a complete description of the syntax and semantics of XML schemas.

```
<?xml version="1.0" encoding="UTF-8"?>
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema" elementFormDefault="qualified"
attributeFormDefault="unqualified">
    <xs:complexType name="ParentType" abstract="true">
        <xs:complexContent>
            <xs:extension base="Child1and2Type">
                <xs:sequence>
                    <xs:element name="Child3">
                        <xs:complexType>
                            <xs:all>
                                <xs:element name="Child3a"/>
                                <xs:element name="Child3b"/>
                            </xs:all>
                        </xs:complexType>
                    </xs:element>
                    <xs:element name="Child4" type="Child4Type" nillable="false"/>
                </xs:sequence>
            </xs:extension>
        </xs:complexContent>
    </xs:complexType>
    <xs:complexType name="Child4Type" abstract="true">
        <xs:sequence>
            <xs:element name="Child4a" type="xs:integer" nillable="false"/>
            <xs:element name="Child4b" type="xs:string" nillable="false"/>
        </xs:sequence>
    </xs:complexType>
    <xs:complexType name="Child1and2Type" abstract="true">
        <xs:sequence>
            <xs:element name="Child2">
                <xs:complexType>
                    <xs:choice maxOccurs="2">
                        <xs:element name="Child2a"/>
                        <xs:element name="Child2b"/>
                        <xs:element name="Child2c"/>
                    </xs:choice>
                </xs:complexType>
            </xs:element>
            <xs:element name="Child1">
                <xs:complexType>
                    <xs:sequence minOccurs="0">
                        <xs:element name="Child1a"/>
                        <xs:element name="Child1b" maxOccurs="unbounded"/>
                        <xs:element name="Child1c" minOccurs="0"/>
                    </xs:sequence>
                </xs:complexType>
            </xs:element>
        </xs:sequence>
    </xs:complexType>
    <xs:element name="Parent" type="ParentType" final="#all"/>
</xs:schema>
```

Figure 2
XML for Sample

7 PDE Schema Definition

7.1 *PDE Class Mapping to XML*

7.1.1 This section describes how the PDE classes from SEMI E139 are mapped to XML. The descriptions in this section are provided to support and explain the XML schema document for the PDE (attached, see ¶7.2.2). This section contains no requirements. Specific requirements are included in ¶7.2.

7.1.2 *Abstract Classes*

7.1.2.1 No classes in the PDE model for RaP are abstract.

7.1.3 *Concrete Classes*

7.1.3.1 This section contains the XML mappings for the concrete classes defined for the PDE in SEMI E139. These are the classes that can be represented in an XML instance document based on the attached XML schema for the PDE.

7.1.4 *PDE*

7.1.4.1 The PDE class is mapped to the XML complexType also named PDE. Table 3 shows how the associations and attributes of the PDE class are mapped into XML. Figure 3 illustrates the resulting XML structure.

7.1.4.2 See ¶7.2.1 for instructions on the calculation of the value of the checksum element.

Table 3 Translation Table For PDE Class

Attribute or Role Name	*UML Type*	*XML Element or Attribute (or Reference)*	*XML Name/Type*
checksum	Checksum	element	checksum (simpleType Checksum)
PDEheader	aggregation	element	PDEheader (complexType)
(xor)	constraint	choice (of PDEbody or PDEbodyReference)	none
PDEbody	aggregation	element	PDEbody (complexType)
PDEbodyReference	aggregation	element	PDEbodyReference (complexType)

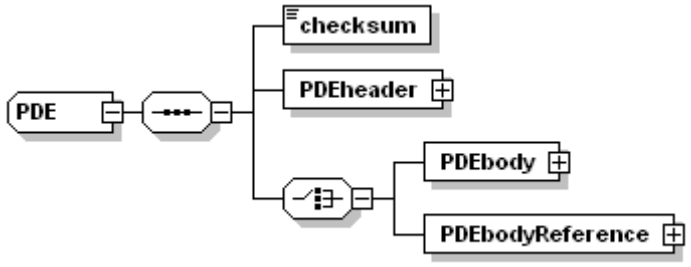

Figure 3
PDE

7.1.4.3 *PDEheader*

7.1.4.3.1 The PDEheader class is mapped to the XML complexType also named PDEheader. Table 4 shows how the associations and attributes of the PDEheader class are mapped into XML. Figure 4 illustrates the resulting XML structure.

Table 4 Translation Table For PDEheader Class

Attribute or Role Name	*UML Type*	*XML Element or Attribute (or Reference)*	*XML Name/Type*
uid	UUID	element	uid (simpleType UUID)
name	String	element	name (simpleType xs:string)
gid	UUID	element	gid (simpleType UUID)
groupName	String	element	groupName (simpleType xs:string)
description	String	element	description (simpleType xs:string)
type	String	element [0..1]	type (simpleType xs:string)
executable	Boolean	element	executable (simpleType xs:boolean)
maxAntecedents	Integer	element	maxAntecedents (simpleType xs:int)
createDate	Time	element	createDate (simpleType xs:dateTime)
createNode	String	element	createNode (simpleType xs:string)
author	String	element	author (simpleType xs:string)

Attribute or Role Name	*UML Type*	*XML Element or Attribute (or Reference)*	*XML Name/Type*
userInfo	list of String	element [0..1]	userInfo (complexType ListOfStrings)
supplierInfo	list of String	element [0..1]	supplierInfo (complexType ListOfStrings)
ExecutionTarget	aggregation	element [0..∞]	ExecutionTarget (complexType)
ReferencedPDE	aggregation	element [0..∞]	ReferencedPDE (complexType)
AntecedentData	aggregation	element [0..∞]	AntecedentData (complexType)
PDEparameter	aggregation	element [0..∞]	PDEparameter (complexType)

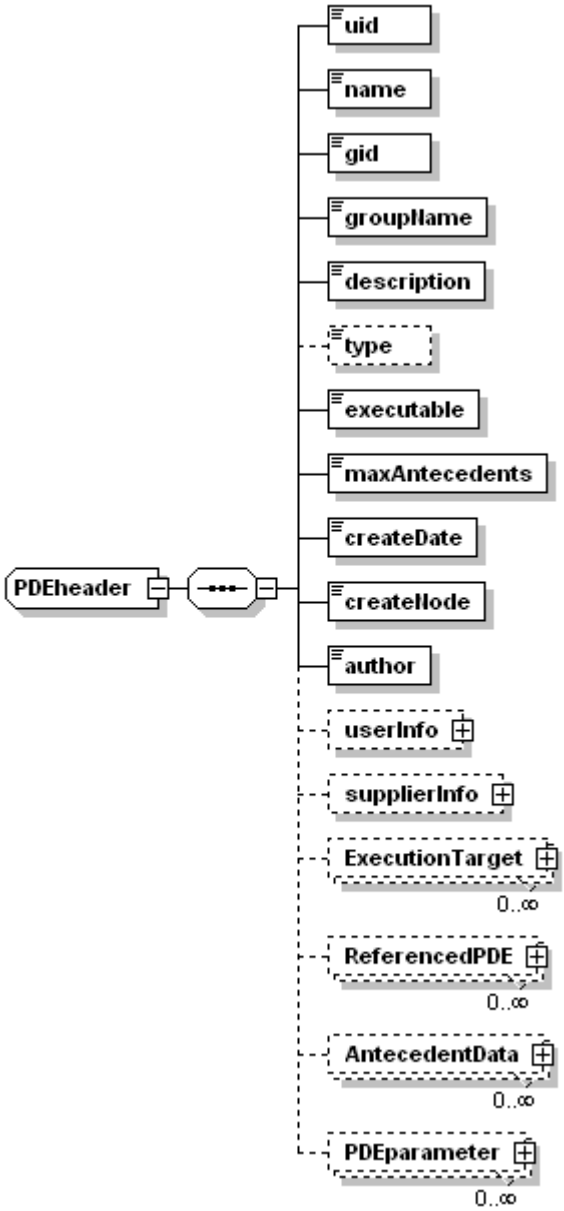

Figure 4
PDEheader

7.1.4.4 *ExecutionTarget*

7.1.4.4.1 The ExecutionTarget class is mapped to the XML complexType also named ExecutionTarget. Table 5 shows how the associations and attributes of the ExecutionTarget class are mapped into XML. Figure 5 illustrates the resulting XML structure.

Table 5 Translation Table For ExecutionTarget

Attribute or Role Name	*UML Type*	*XML Element or Attribute (or Reference)*	*XML Name/Type*
identifier	String	element [0..1]	identifier (simpleType xs:string)
supplier	String	element	supplier (simpleType xs:string)
make	String	element	make (simpleType xs:string)
model	String	element	model (simpleType xs:string)
recipeTypes	list of String	element [0..1]	recipeTypes (complexType ListOfStrings)

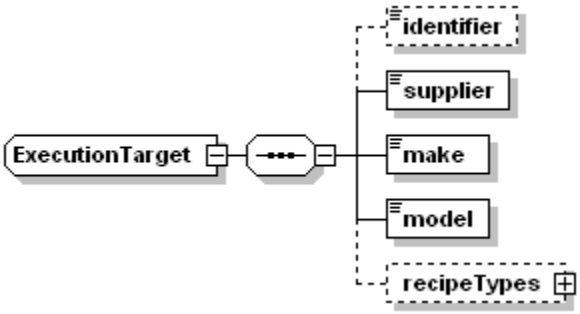

Figure 5
ExecutionTarget

7.1.4.5 *ReferencedPDE*

7.1.4.5.1 The ReferencedPDE class is mapped to the XML complexType also named ReferencedPDE. Table 6 shows how the associations and attributes of the ReferencedPDE class are mapped into XML. Figure 6 illustrates the resulting XML structure.

Table 6 Translation Table For ReferencedPDE

Attribute or Role Name	*UML Type*	*XML Element or Attribute (or Reference)*	*XML Name/Type*
id	UUID	element	id (simpleType UUID)

Figure 6
ReferencedPDE

7.1.4.6 *AntecedentData*

7.1.4.6.1 The AntecedentData class is mapped to the XML complexType also named AntecedentData. Table 7 shows how the associations and attributes of the ReferencedPDE class are mapped into XML. Figure 7 illustrates the resulting XML structure.

Table 7 Translation Table For AntecedentData

Attribute or Role Name	*UML Type*	*XML Element or Attribute (or Reference)*	*XML Name/Type*
uid	UUID	element	uid (simpleType UUID)
name	String	element	name (simpleType xs:string)
gid	UUID	element	gid (simpleType UUID)
groupName	String	element	groupName (simpleType xs:string)
description	String	element	description (simpleType xs:string)
author	String	element	author (simpleType xs:string)
createDate	Time	element	createDate (simpleType xs:dateTime)
createNode	String	element	createNode (simpleType xs:string)
AntecedentData	aggregation	element [0..∞]	AntecedentData (complexType)

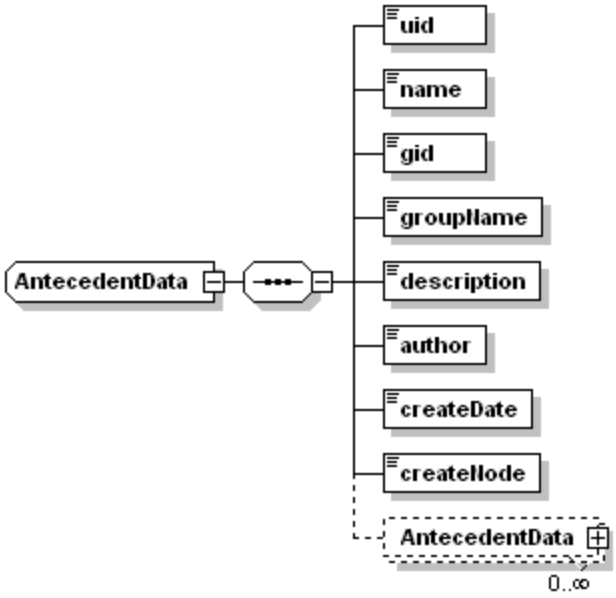

Figure 7
AntecedentData

7.1.4.7 *PDEparameter*

7.1.4.7.1 The PDEparameter class is mapped to the XML complexType also named PDEparameter. Table 8 shows how the attributes of PDEparameter are mapped to XML. The resulting XML structure is illustrated in Figure 8.

Table 8 Translation Table For PDEparameter

Attribute or Role Name	*UML Type*	*XML Element or Attribute (or Reference)*	*XML Name/Type*
name	String	element	name (simpleType xs:string)
description	String	element	description (simpleType xs:string)
units	String	element	units (simpleType xs:string)
relatedParameters	list of String	element [0..1]	relatedParameters (complexType ListOfStrings)
defaultValue	Any	element [0..1]	defaultValue (complexType xs:anyType)
inputBoundaryType	Enumeration	element [0..1]	inputBoundaryType (enumeration inputBoundsType)
inputBounds	Any	element [0..∞]	inputBounds (complexType xs:anyType)

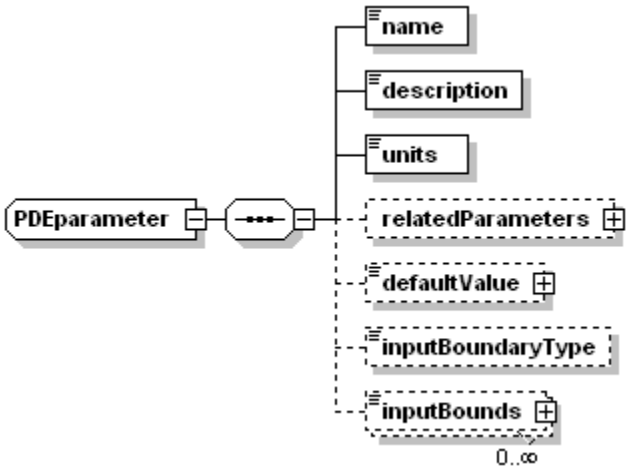

Figure 8
PDEparameter

7.1.4.8 *PDEbody*

7.1.4.8.1 The PDEbody class is mapped to the XML complexType also named PDEbody. Table 9 shows that the PDEbody is of type xs:any. This type allows the implementation to place elements and attributes in an instance of PDEbody without restriction. The resulting XML structure is illustrated in Figure 9.

Table 9 Translation Table For PDEbody

Attribute or Role Name	*UML Type*	*XML Element or Attribute (or Reference)*	*XML Name/Type*
none	n/a	element or attribute [0..∞]	user defined

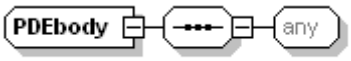

Figure 9
PDEbody

7.1.4.9 *PDEbodyReference*

7.1.4.9.1 The PDEbodyReference class is mapped to the XML complexType also named PDEbodyReference. Table 10 shows how the attributes of PDEbodyReference are mapped to XML. The resulting XML structure is illustrated in Figure 10.

7.1.4.9.2 See ¶7.2.1 for instructions on the calculation of the value of the bodyChecksum element.

Table 10 Translation Table For PDEbodyReference

Attribute or Role Name	*UML Type*	*XML Element or Attribute (or Reference)*	*XML Name/Type*
bodyChecksum	Checksum	element	bodyChecksum (simpleType Checksum)
specification	String	element	specification (simpleType xs:string)

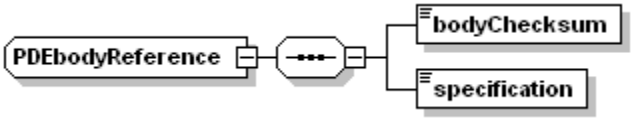

Figure 10
PDEbodyReference

7.1.5 *Added XML Constructs*

7.1.5.1 In the process of translating the UML model for the PDE into a useful XML schema, is was necessary to add certain types and constructs to the schema. This section describes these XML constructs.

7.1.5.2 *XML simpleTypes*

7.1.5.2.1 Table 11 describes each simpleType[2] added to the PDE schema.

Table 11 simpleTypes In The PDE Schema

Type	***Description***
UUID	Holds a standard 36-character uuid string. The pattern is defined as five groups of characters, each separated by the "-" character. Here is an example value in UUID format: "44BBA855-CC51-11CF-AAFA-00AA00B6015C".
Checksum	Holds a 128-bit MD5 checksum value. The simpleType is hexBinary – 32 characters representing 4 bits each (16 bytes).
InputBoundsType	This construct defines the enumerated choices (List or Range) needed for the attribute inputBoundsType of PDEparameter.

7.1.5.3 *Added XML complexTypes*

7.1.5.3.1 Table 12 describes each complexType[2] added to the PDE schema.

Table 12 complexTypes In The PDE Schema

Type	***Description***
ListOfStrings	Holds a collection of zero or more strings.

7.2 *Requirements For PDE Schema*

7.2.1 This section contains all requirements specified for the PDE Schema. ¶7.1 above is included for documentation purposes.

7.2.2 *Checksum Calculation*

7.2.2.1 The method for calculation of checksum values is specified by the SEMI E139 RaP standard to be MD5. The MD5 calculation processes a sequence of bytes (or octets) to yield the checksum value. This section specifies that sequence of bytes for the PDE and external PDEbody.

7.2.2.2 *PDE Checksum*

7.2.2.2.1 The PDE *checksum* element value shall be calculated based on the XML instance document for the PDE. This applies to PDEs that have internal PDEbodies and also to those that do not.

7.2.2.2.2 XML documents can contain white space and layout information that is independent of the XML content. So, two XML documents that have identical XML content might not be identical byte for byte. Allowance has been made for these differences.

7.2.2.2.3 The W3C recommendation xml-c14n (http://www.w3c.org/TR/xml-c14n) defines a method for creating a canonical form of an XML document. This canonicalization process eliminates the white space and layout information from the checksum calculation. The xml-c14n recommendation shall be applied in the following way:

- The checksum calculation shall be performed on a canonicalized form of the RaP:PDE element. The computation of the checksum shall be equivalent to the procedure described below:
  - In the RaP:PDE element of the original PDE replace the value of the 'checksum' element by 00000000000000000000000000000000 (that is, 32 zeroes).

2 "simpleType" and "complexType" are XML terms (see also "xs:simpleType" and "xs:complexType").

- Convert the modified PDE element to its canonical form according to W3C recommendation xml-c14n without comments (that is, the "with comments" option is not used). This yields an octet stream encoded in UTF-8.
- Apply the MD5 algorithm to the resulting octet stream to compute the checksum.

7.2.2.2.4 The resulting checksum value can be used as needed. For example, if a new PDE is being created, the value may be placed into the checksum element. For an existing PDE, the value may be compared with the existing value of the checksum element to determine if the content is unchanged.

7.2.2.3 *PDEbody BodyChecksum*

7.2.2.3.1 The format of an external PDEbody is determined by the equipment supplier. In this case, the entire PDEbody content as contained within TransferContainer shall be included in the checksum calculation process. The calculation method for the bodyChecksum attribute of the PDEbodyReference element is MD5 as specified by the E139 RaP standard.

7.2.3 *XML Schema*

7.2.3.1 All PDE instance documents shall conform to the XML schema document named:

E139-1.V0705.RaP.PDE.xsd

7.2.3.2 The contents of the above-mentioned schema document constitute a core part of this specification. This schema document should be provided with this document.

7.2.3.3 In addition, all PDE instance documents shall conform to all requirements related to the PDE content as specified in SEMI E139.

8 Manifest Schema Definition

8.1 *Manifest Mapping to XML*

8.1.1 This section describes how the Manifest definition from SEMI E139 is mapped to XML. The descriptions in this section are provided to support and explain the XML schema document (attached, see ¶8.2). This section contains no requirements. Specific requirements are included in ¶8.2.

8.1.2 *Manifest Class*

8.1.2.1 SEMI E139 does not represent the Manifest in UML. To facilitate creation of the Manifest schema, the Manifest itself is treated as a single class containing an unordered sequence of Entries.

8.1.2.2 The first two columns of the Translation Table for this Manifest class (Table 13) hold little information because this class was not explicitly defined in SEMI E139. However, the second two columns describe the XML form of the Manifest. Figure 11 shows the XML form of the Manifest.

Table 13 Translation Table For Manifest Class

Attribute or Role Name	*UML Type*	*XML Element or Attribute (or Reference)*	*XML Name/Type*
<none>	<none>	element	Manifest (complexType)
<none>	<none>	element	Entry (complexType)
<none>	UUID	element	uid (simpleType UUID)
<none>	String	element	PDEdescriptor (simpleType xs:string)
<none>	String	element	PDEbodyDescriptor (simpleType xs:string)
<none>	String	element	location (simpleType xs:string)

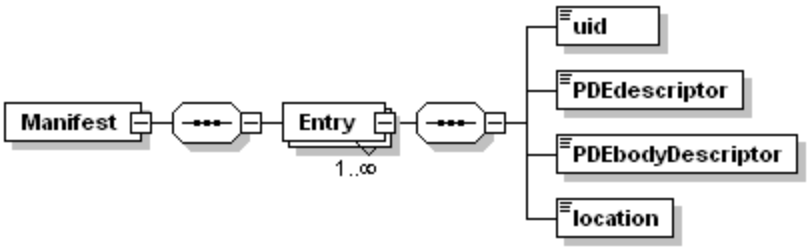

Figure 11
Manifest

8.1.3 *XML simpleTypes*

8.1.3.1 Table 11 describes each simpleType added to the XML schema for the Manifest.

Table 14 simpleTypes In The PDE Schema

Type	*Description*
UUID	Holds a standard 36-character uuid string. The pattern is defined as five groups of characters, each separated by the "-" character. Here is an example value in UUID format: "44BBA855-CC51-11CF-AAFA-00AA00B6015C".

8.2 *Requirements For Manifest Schema*

8.2.1 This section contains all requirements specified for the PDE Schema. ¶8.1 above is included for documentation purposes.

8.2.2 All Manifest instance documents shall conform to the XML schema document named:

E139-1.V0705.RaP.Manifest.xsd

8.2.3 The contents of the above-mentioned schema document constitute a core part of this specification. This schema document should be provided with this document.

8.2.4 In addition, all Manifest instance documents shall conform to all requirements related to the Manifest content as specified in SEMI E139.

9 Related Documents

Rivest, Ronald, *The MD5 Message-Digest Algorithm*, IETF RFC: 1992 http://www.ietf.org/rfc/rfc1321.txt.

SEMI E142-0705
SPECIFICATION FOR SUBSTRATE MAPPING

This specification was technically approved by the Global Information & Control Committee and is the direct responsibility of the North American Information & Control Committee. Current edition approved by the North American Regional Standards Committee on December 10, 2004. Initially available at www.semi.org January 2005; to be published March 2005.

NOTICE: The designation of SEMI M55 was updated during the 0705 publishing cycle to reflect the creation of SEMI E142.1.

1 Purpose

1.1 This document defines the data items that are required to report, store and transmit map data for substrates such as wafers, frames, strips and trays.

2 Scope

2.1 This version of the document applies to the substrate types; wafers, frames, strips and trays.

2.2 This document addresses assembly and packaging including the testing of semiconductor devices.

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

3 Limitations

3.1 This document does not address the transmission, file naming conventions, storage or archiving of substrate maps. These will be addressed in sub-documents of this specification.

4 Referenced Standards

4.1 *SEMI Standards*

SEMI E39 — Object Services Standard; Concept, Behavior and Services

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

5 Terminology

5.1 *Definitions*

5.1.1 *bottom side* — the bottom side of the substrate as defined in the corresponding Appendix (Appendix 1, 2, or 3).

5.1.2 *device* — the unit to which the device status code in the map is assigned including, but not limited to: die on a wafer, multi-chip modules, and packages.

5.1.3 *map* — a two-dimensional array of bin codes derived from electrical test data of a substrate including, but not limited to: wafer, tray, strip, or tape.

5.1.4 *substrate* — any carrier of a two-dimensional array of devices including, but not limited to: wafers, trays, strips, tape, panels, or boards.

5.1.5 *top side* — the top side of the substrate as defined in the corresponding Appendix for that substrate (Appendix 1, 2, or 3).

6 Requirements

6.1 This standard defines the data items that are required in a substrate map. This standard places no special requirements on the file naming conventions, storage, or archiving of substrate maps.

7 Conventions

7.1 *Object Conventions* — This document conforms to the conventions for objects established by SEMI E39, including object diagrams, object terminology, and requirements for standardized objects. Accordingly, notation is based on Unified Modeling Language (UML).

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

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

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

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

7.2.2.1 The Form column is used to indicate the format of the attribute.

7.2.3 *Formal Name of an Object* — The text capitalizes formal object name references. Similar to the way capitalization is normally used when discussing entities. When describing something in the general (like cities) lower case is used, but when a specific entity is of interest (New York City), then first letters are capitalized.

7.2.4 *Components of Complex Attributes* — The names of object attributes defined in tables are left-justified. The individual elements of complex attributes are right-justified in order of appearance below the complex attribute.

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

7.3.1 *Service Definition*

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

Message Service Name	*Type*	*Description*

7.3.1.2 Type can be either "N" = Notification or "R" = Request & Response.

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

7.3.2 *Service Parameter Dictionary*

7.3.2.1 A service parameter dictionary table defines the description, format and its possible value for parameters used by services, as shown in the following table:

Parameter Name	*Description*	*Format: Possible Value*

7.3.2.2 A row is provided in the table for each parameter of a service.

7.3.3 *Service Message Definition*

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

Parameter	*Req/Ind*	*Rsp/Cnf*	*Comment*

7.3.3.2 The columns labeled REQ/IND and RSP/CNF link the parameters to the direction of the message. The message sent by the initiator is called the "Request". The receiver terms this message the "Indication" or the request. The receiver may then send a "Response" which the original sender terms the "Confirmation".

7.3.3.3 The following codes appear in the REQ/IND and RSP/CNF columns and are used in the definition of the parameters (eg., how each parameter is used in each direction):

M	Mandatory Parameter — Must be given a valid value.
C	Conditional Parameter — May be defined in some circumstances and undefined in others. Whether a value is given may be completely optional or may depend on the value of the other parameter.
U	User-Defined Parameter.
-	The parameter is not used.
=	(For response only.) Indicates that the value of this parameter in the response must match that in the primary (if defined).

8 Overview

8.1 Substrate maps are two dimensional arrays of data that correspond to a physical substrate which may be a wafer, strip or tray as shown in the Appendices. It is expected that other substrate types will be added to subsequent releases of this standard.

8.2 The data items are defined as attributes of the objects shown in Figure 1 Map Data Object Model.

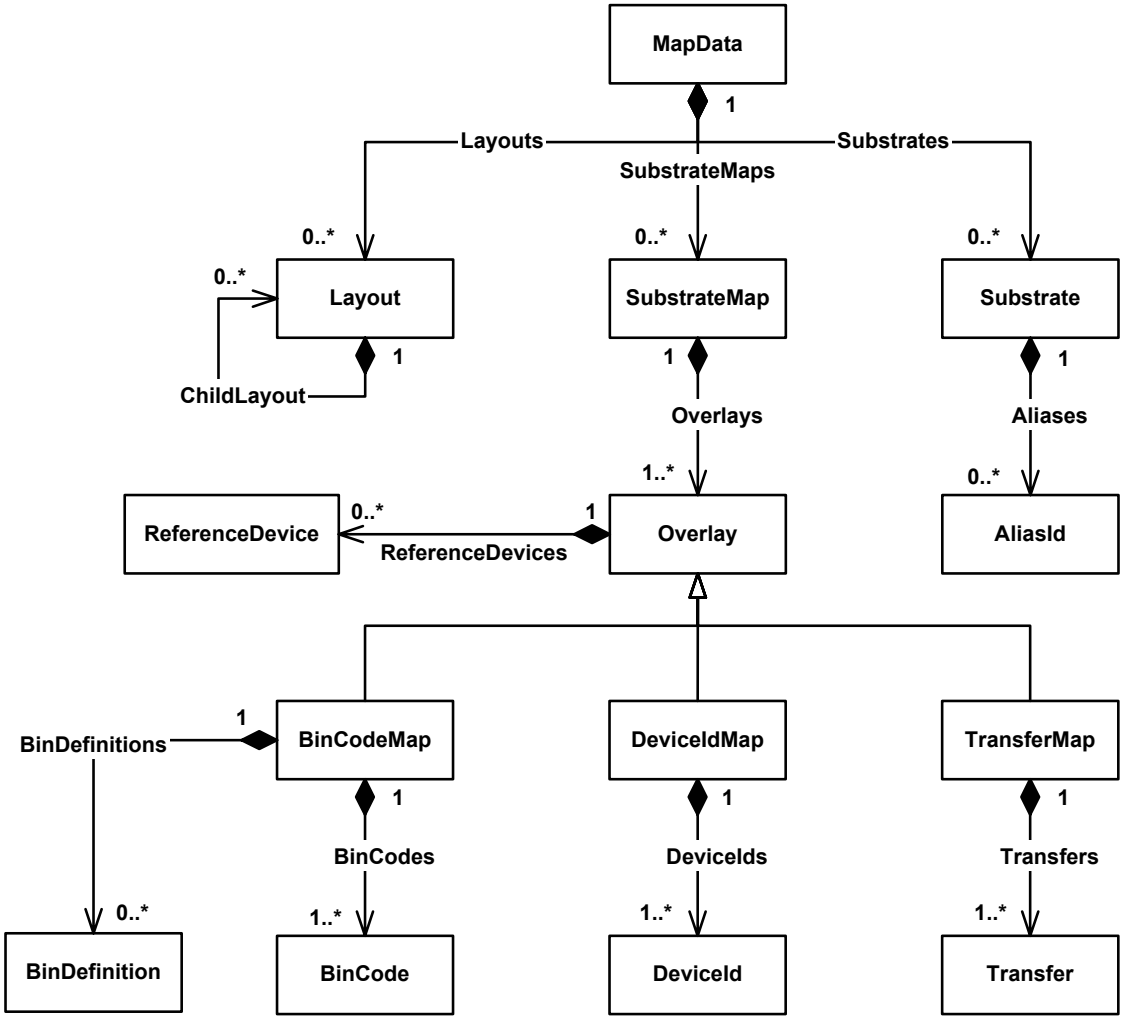

Figure 1
Map Data Object Model

9 MapData

9.1 The MapData object contains a list of zero or more Layout objects, a list of zero or more SubstrateMap objects and a list of zero or more Substrate objects.

Table 1 MapData Attributes

Attribute Name	Definition	Access	Reqd	Form
FormatRevision	Specifies the exact name and revision of the format used to represent substrate map data. This field can be used by a parser to automatically determine the format of the remaining substrate map data.	RO	N	Text

10 Layout

10.1 The Layout object defines the logical and physical dimensions of two-dimensional array of devices to which a SubstrateMap may be assigned. A substrate must have a top level layout that defines the size of the substrate itself. Any layout object can reference other child layout objects. In this way complex assemblies may be represented. See the examples in Related Information 1.

Table 2 Layout Attributes

Attribute Name	*Definition*	*Access*	*Reqd*	*Form*
LayoutId	The layout identifier. This must be unique within the scope of MapData.	RO	Y	Text
DefaultUnits	The units in which physical dimensions and coordinates are represented unless otherwise specified.	RO	Y	Text
Dimension	The number of columns (X-axis) and rows (Y-axis) in the layout. If this is the top level layout then X = 1, Y = 1.	RO	Y	LogicalCoordinates (see Table 3)
DeviceSize	The X and Y dimensions of the device. If this is the top level layout, then this defines the size of the substrate itself. If not present, then DeviceSize is not defined.	RO	N	XYDimensions (see Table 4)
StepSize	The distance from a reference point on one device to the same reference point on the adjacent device. StepSize – DeviceSize is equal to the "street" width between devices. If not present, then StepSize is not defined.	RO	N	XYDimensions (see Table 4)
LowerLeft	The distance in microns of the lower left corner of this layout from the lower left corner of the parent layout. If this is the top level layout, then LowerLeft.X = 0 and LowerLeft.X = 0. If not present, then LowerLeft.X = 0 and LowerLeft.X = 0.	RO	N	XYDimensions (see Table 4)
Z	The logical and physical height of the device above (or below) the parent layout. If this is the top level layout, then Z.Order = 0 and Z.Height = 0. If not present, then Z.Order = 0 and Z.Height = 0.	RO	N	ZDimensions (see Table 5)
TopImage	Name of an image file showing the top side of an individual device in the layout.	RO	N	Text
BottomImage	Name of an image file showing the bottom side of an individual device in the layout.	RO	N	Text
ProductId	Product identifier for a device in this layout.	RO	N	Text
TopLevel	TopLevel = True implies that this is a top level layout. TopLevel = False implies that this layout is the child of another layout. If this is the top level layout, then TopLevel = True. If not present, then TopLevel = False. TopLevel may be omitted from child layouts.	RO	N	Boolean

Table 3 LogicalCoordinates Type

Attribute Name	*Definition*	*Access*	*Reqd*	*Form*
X	The column or X logical coordinate, according to the values of SubstrateMap coordinate attributes; SubstrateSide, Orientation, OriginLocation, and AxisDirection.	RO	Y	Integer
Y	The row or Y logical coordinate, according to the values of SubstrateMap coordinate attributes; SubstrateSide, Orientation, OriginLocation, and AxisDirection.	RO	Y	Integer

Table 4 XYDimensions Type

Attribute Name	*Definition*	*Access*	*Reqd*	*Form*
X	The physical offset from the origin along the X axis, according to the values of SubstrateMap coordinate attributes; SubstrateSide, Orientation, OriginLocation, and AxisDirection.	RO	Y	Floating point value
Y	The physical offset from the origin along the Y axis, according to the values of SubstrateMap coordinate attributes; SubstrateSide, Orientation, OriginLocation, and AxisDirection.	RO	Y	Floating point value
Units	The units in which X and Y coordinates are represented. If omitted then the Layout, DefaultUnits is used.	RO	N	Text

Table 5 ZDimensions Type

Attribute Name	*Definition*	*Access*	*Reqd*	*Form*
Order	The order of the layouts in the Z dimension, according to the values of SubstrateMap coordinate attributes; SubstrateSide, and Orientation. Layouts with higher values for Order are in front of those with smaller values. If this is the top level layout, then Order =0. If not present, then Order =0.	RO	N	Integer
Height	The height of the device above (or below) the parent layout, according to the values of SubstrateMap coordinate attributes; SubstrateSide, and Orientation. If this is the top level layout, then Height =0. If not present, then Height =0.	RO	N	Floating point value
Units	The units in which Height is represented. If omitted then the Layout.DefaultUnits is used.	RO	N	Text

11 Substrate

11.1 The Substrate object identifies a substrate to which a SubstrateMap may be assigned. A Substrate object may have one or more AliasId objects.

Table 6 Substrate Attributes

Attribute Name	*Definition*	*Access*	*Reqd*	*Form*
SubstrateType	The type of substrate. A choice of ‘Wafer’, ‘Frame’, ‘Strip” or ‘Tray”.	RO	Y	Text
SubstrateId	The substrate identifier.	RO	Y	Text 1 to 32 characters
LotId	Production lot identifier for this data.	RO	N	Text 1 to 32 characters
CarrierType	The type of wafer carrier, e.g. ‘Cassette’, ‘FOUP’, etc.	RO	N	Text
CarrierId	A character code that may be printed or encoded on the substrate carrier, which uniquely identifies the substrate carrier.	RO	N	Text 1 to 32 characters
SlotNumber	This identifies the slot in the carrier in which the substrate is placed.	RO	N	Positive Integer
SubstrateNumber	Identifies the substrate within a lot.	RO	N	Positive Integer

Attribute Name	*Definition*	*Access*	*Reqd*	*Form*
GoodDevices	The total number of “good” devices on the substrate. The definition of “good” is beyond the scope of this document.	RO	N	Positive Integer
SupplierName	Name of the supplier of the substrate.	RO	N	Text
CreateDate	Date and time when the map data is acquired: formatted as YYYYMMDDhhmmsscc (year-month-date-hour-minute-second-centisecond).	RO	N	Text
LastModified	Date and time when the map data was last modified: formatted as YYYYMMDDhhmmsscc (year-month-date-hour-minute-second-centisecond).	RO	N	Text
Status	Status of map data supplying process or preceding process.	RO	N	Text

11.2 *AliasId*

11.2.1 The AliasId object contains other substrate identifiers that may be used to trace the substrate throughout the substrate’s life cycle.

Table 7 AliasId Attributes

Attribute Name	*Definition*	*Access*	*Reqd*	*Form*
Type	Type of identifier, e.g. “FrontsideId”, “BacksideId”, “WaferId”, “FrameId” .	RO	Y	Text
Value	Value of the identifier.	RO	Y	Text 1 to 32 characters

12 SubstrateMap

12.1 The SubstrateMap object relates to a single substrate and layout and contains a list of one or more Overlay objects. It may also include a substrate type specific object depending on the value of [SubstrateType].

12.2 If there are more than one SubstrateMap assigned to a Substrate and Layout combination they should differ in either their Orientation, OriginLocation or both. For example there may be a map for the top side and one for the bottom side of a Layout.

Table 8 SubstrateMap Attributes

Attribute Name	*Definition*	*Access*	*Reqd*	*Form*
SubstrateType	A reference to the type of substrate to which the map data applies.	RO	Y	Text
SubstrateId	A reference to the substrate identifier to which the map data applies.	RO	Y	Text 1 to 32 characters
LayoutSpecifier	A reference to the layout to which the map data applies. The LayoutSpecifier defines the full path, using / as a delimiter from the top level layout to the layout to be specified for example; ”WaferLayout/L1/DeviceLayout”.	RO	Y	Text
SubstrateSide	Coordinate system attribute that defines the side of the substrate. Restricted to the values; TopSide* Bottom Side * Default value if this item is not present.	RO	N	Text

Attribute Name	*Definition*	*Access*	*Reqd*	*Form*
Orientation	Coordinate system attribute that defines the orientation of the substrate in relation to the map data. This variable will increase in the right (clockwise) direction from 0°. It is restricted to values from 0°-359°. * Default value if this item is not present is 0°..	RO	N	Integer
OriginLocation	Coordinate system attribute that defines location of the coordinate system origin. Restricted to the values (see Figure A2-1 for an example); LowerLeft* UpperLeft LowerRight UpperRight Center * Default value if this item is not present.	RO	N	Text
AxisDirection	Coordinate system attribute that defines direction in which the Y and X axis, respectively increase. Restricted to the values (see Figure A2-1 for an example); UpRight* DownRight UpLeft DownLeft * Default value if this item is not present.	RO	N	Text

13 Overlay

13.1 The Overlay object is an array of data that corresponds to the dimensions on the Layout to which the containing SubstrateMap is assigned. The specialized objects (e.g. BinCodeMap, DeviceIdMap, TransferMap) are extensions of the Overlay object which each define a different type of data. See the definitions of the specialized objects in the sections below. An Overlay object may have one or more ReferenceDevice objects.

Table 9 Overlay Attributes

Attribute Name	*Definition*	*Access*	*Reqd*	*Form*
MapName	A name that describes the purpose of the bin codes in the map. Possible examples include: “CellStatus”, “DefectCode”, “MarkGrade”, “PackageGrade”, “SortGrade”, DeviceId”, “Transfer”.	RO	Y	Text
MapVersion	The version is used to distinguish between multiple versions of maps for the same substrate with the same MapName.	RO	N	Text

13.2 *ReferenceDevice*

13.2.1 The ReferenceDevice object defines a reference device on the substrate.

Table 10 ReferenceDevice Attributes

Attribute Name	*Definition*	*Access*	*Reqd*	*Form*
Coordinates	X and Y coordinate of the reference device to align device matrix on physical substrate with the map data.	RO	Y	LogicalCoordinates (see Table 3)
Position	Offset of the center of a reference device in the X and Y directions from the center of substrate.	RO	N	XYDimensions (see Table 4)

Attribute Name	*Definition*	*Access*	*Reqd*	*Form*
Name	Descriptive name for the reference, e.g. “First Device”, “Prober Origin”	RO	N	Text

13.3 *BinCodeMap*

13.3.1 The BinCodeMap object contains a list of zero or more BinDefinition objects and a list of one or more BinCode objects.

Table 11 BinCodeMap Attributes

Attribute Name	*Definition*	*Access*	*Reqd*	*Form*
BinType[#1, #2]	The format in which the each device will be represented. The following values are reserved: BinType = ‘Ascii’: Single ASCII character (1 byte per device) BinType = ‘Decimal’: 3 digit integer from 000 to 255 (1 byte per device) BinType = ‘HexaDecimal’: 2 digit Hexadecimal value from 00 to FF (1 byte per device) BinType = ‘Integer2’: 4 digit Hexadecimal value from 0000 to FFFF (2 bytes per device)	RO	Y	Text
NullBin	Code to indicate no device or not measured device in BinCode.Values. It should be represented according to BinType.	RO	Y	Text

[#1] For readability when BinType = ‘Decimal’ there must be spaces added between each device and each row of devices should be on a new line.

[#2] When BinType = ‘Ascii’ or ‘Hexadecimal’ or ‘Integer2’ there must not be spaces between each device and for readability, each row should be on a new line.

13.3.2 *BinCode*

13.3.2.1 The BinCode contains the actual bin code values.

Table 12 BinCode Attributes

Attribute Name	*Definition*	*Access*	*Reqd*	*Form*
Values	The concatentation of bin code values represented according to the value of BinCodeMap.BinType. [Values] may contain just one bin code, a row of bin codes or the entire map of bin codes.	RO	Y	Text
X	The X coordinate of the device to which the first bin code applies, according to the values of SubstrateMap coordinate attributes; SubstrateSide, Orientation, OriginLocation, and AxisDirection. If omitted X = 0.	RO	N	Integer
Y	The Y coordinate of the device to which the first bin code applies, according to the values of SubstrateMap coordinate attributes; SubstrateSide, Orientation, OriginLocation, and AxisDirection. If omitted Y is derived from the ordinal position of the BinCode object in the parent BinCodeMap object. The first BinCode object is the top row.	RO	N	Integer

Attribute Name	*Definition*	*Access*	*Reqd*	*Form*
Number	The number of contiguous bin codes in the Values attribute. If omitted Number is derived from the length of the Values attribute and the value of BinCodeMap.BinType. For example, if BinCodeMap.BinType = 'HexaDecimal', Number = LengthOf(Values) / 2	RO	N	Integer

13.3.3 *Map Representation Types*

13.3.3.1 A BinCode object may contains bin code information for a single device, a row of devices or the entire map of devices. The choice depends on whether the primary objective is readability, compactness or flexibility.

13.3.3.2 The most readable approach is to represent all the devices in a row in a single BinCode entry. The length of each row is equal to the value of Layout.Columns. Locations in the layout that do not contain a device are assigned to the BinCodeMap.NullBin value. Alternatively, a starting XY coordinate and length may be specified. This approach offers good readability and a balance between compactness and flexibility. See Figure A1-1a Row/Column Format.

13.3.3.3 For a more compact representation, a single BinCode object can include the entire array of bin codes in row major format starting at the top left. See Figure A1-1b Array Format.

13.3.3.4 A less compact but more flexible approach is to include a separate BinCode object for each individual device. The X and Y coordinates must both be included. See Figure A1-1c Coordinate Format

13.3.4 *BinDefinition*

13.3.4.1 The BinDefinition object describes a bin code that may be present in the map.

Table 13 BinDefinition Attributes

Attribute Name	*Definition*	*Access*	*Reqd*	*Form*
BinCode	A bin category, other that the value assigned to BinCodeMap.NullBin, that may be assigned to a device. It should be represented according to BinCodeMap.BinType.	RO	Y	Unsigned 1 byte integer unless BinCodeMap.BinType attribute is 'Integer2', in which case it is a 2 byte integer.
BinCount	The number of devices on the substrate with the specified BinCode. If not present then BinCount is undefined.	RO	N	Positive Integer
BinDescription	A description of the specified BinCode, e.g. "100MHz". If not present then BinDescription is undefined.	RO	N	Text
BinQuality	Describes the quality of the specified BinCode. The following values are reserved: "Pass" – Indicates a quality that has commercial value "Fail" – Indicates a quality that does not have commercial value Other values defined by an application. If not present then BinQuality is undefined.	RO	N	Text
Pick	Pick = True indicates that devices with this BinCode should be processed. Pick = False indicates that devices with this BinCode should not be processed. If not present that Pick=False.	RO	N	Boolean

13.4 *DeviceIdMap*

13.4.1 The DeviceIdMap object contains a list of one or more DeviceId objects.

13.4.2 *DeviceId*

13.4.2.1 The DeviceId object contains the device coordinate and the value of the identifier marked on that device.

Table 14 DeviceId Attributes

Attribute Name	*Definition*	*Access*	*Reqd*	*Form*
Value	The device identifier.	RO	Y	Text
X	The X coordinate of the device with the substrate oriented, according to the values of SubstrateMap coordinate attributes; SubstrateSide, Orientation, OriginLocation, and AxisDirection.	RO	Y	Integer
Y	The Y coordinate of the device with the substrate oriented, according to the values of SubstrateMap coordinate attributes; SubstrateSide, Orientation, OriginLocation, and AxisDirection.	RO	Y	Integer

13.5 *TransferMap*

13.5.1 The TransferMap object contains a list of one or more Transfer objects.

Table 15 TransferMap Attributes

Attribute Name	*Definition*	*Access*	*Reqd*	*Form*
FromSubstrateType	The type of substrate from which the devices were transferred. A choice of 'Wafer', 'Frame', 'Strip" or 'Tray".	RO	Y	Text
FromSubstrateId	The identifier of substrate from which the devices were transferred.	RO	Y	Text

13.5.2 *Transfer*

13.5.2.1 The Transfer object defines, for an individual device, the coordinates on the substrate from which the device was picked and the coordinates at which the device was placed on the substrate to which this TransferMap applies.

Table 16 Transfer Attributes

Attribute Name	*Definition*	*Access*	*Reqd*	*Form*
FX	The X coordinate on the substrate from which the device was picked, according to the values of SubstrateMap coordinate attributes; SubstrateSide, Orientation, OriginLocation, and AxisDirection.	RO	Y	Integer
FY	The Y coordinate on the substrate from which the device was picked, according to the values of SubstrateMap coordinate attributes; SubstrateSide, Orientation, OriginLocation, and AxisDirection.	RO	Y	Integer
TX	The X coordinate at which the device was placed on the substrate to which this map applies, according to the values of SubstrateMap coordinate attributes; SubstrateSide, Orientation, OriginLocation, and AxisDirection.	RO	Y	Integer

Attribute Name	*Definition*	*Access*	*Reqd*	*Form*
TY	The Y coordinate at which the device was placed on the substrate to which this map applies, according to the values of SubstrateMap coordinate attributes; SubstrateSide, Orientation, OriginLocation, and AxisDirection.	RO	Y	Integer

14 Services

14.1 Table 18 shows the services defined for map data.

Table 17 Map Data Services

Message Service Name	*Type*	*Description*
MapDownload	R	Download Map Data
MapUpload	N	Upload Map Data

14.2 *Map Download Service*

14.2.1 This service may be used to download MapData for a single substrate.

Table 18 Map Download Parameter Dictionary

Parameter Name	*Description*	*Format: Possible values*
SubstrateType	The type of substrate to which the MapData applies.	Text Restricted to the values: ‘Wafer’, ‘Frame’, ‘Strip” or ‘Tray”
SubstrateId	The identifier of the substrate to which the MapData applies.	Text
MapData	The MapData for the specified substrate.	Text Other standards that define an implementation of this service and reference this standard will define how the MapData is represented in text.

Table 19 Map Download Service Definition

Parameter	*Req/Ind*	*Rsp/Cnf*	*Comment*
SubstrateType	M	-	Type of substrate to which download.
SubstrateId	M	-	Identifier of substrate to download.
MapData	-	M	The MapData for the specified substrate.

14.3 *Map Upload Service*

14.3.1 This service may be used to upload MapData for one ore more substrates.