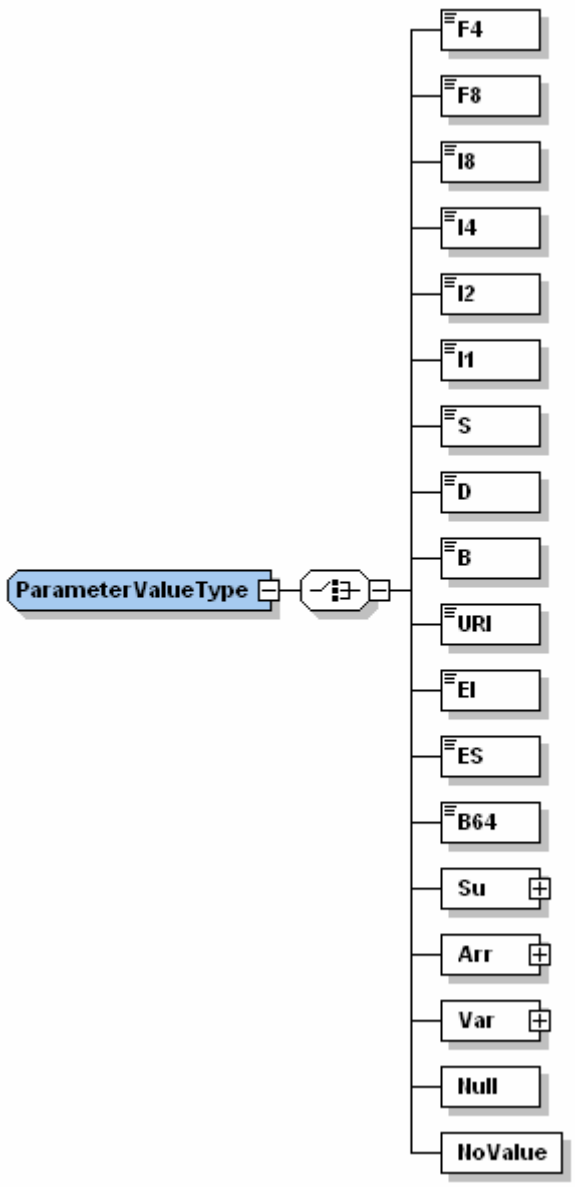

Figure 28
ParameterValueType

Table 75 Translation Table for ParameterValueType

Attribute or Role Name	*UML Type*	*XML Element or Attribute (or Reference)*	*XML Name/Type*
value	RealValue	Element	F4: xsd:float
value	RealValue	Element	F8: xsd:double
value	IntegerValue	Element	I8: xsd:long
value	IntegerValue	Element	I4: xsd:int
value	IntegerValue	Element	I2: xsd:short
value	IntegerValue	Element	I1: xsd:byte
value	StringValue	Element	S: xsd:string

Attribute or Role Name	*UML Type*	*XML Element or Attribute (or Reference)*	*XML Name/Type*
value	SEMI E134.1 Extension	Element	D: xsd:dateTime
value	BooleanValue	Element	B: xsd:boolean
value	SEMI E134.1 Extension	Element	URI: xsd:anyURI
value	EnumeratedIntegerValue	Element	EI: xsd:int
value	EnumeratedStringValue	Element	ES: xsd:string
value	BinaryValue	Element	B64: xsd:base64Binary
value	StructureValue	Element	Su: dcm: CompositeValueType
value	ArrayValue	Element	Arr: dcm: CompositeValueType
value	SEMI E134.1 Extension	Element	Var: dcm: CompositeValueType
SEMI E134.1 Extension	SEMI E134.1 Extension	Element	Null: dcm:NullType
value	NoValue	Element	NoValue: dcm:NoValueType

7.2.18.4.5 *SEMI E125.1 Parameter to SEMI E134.1 Parameter Mapping* — If a requested Parameter has an SEMI E125.1 type definition, then suppliers shall communicate their values according to the following table. The following table describes the mapping of SEMI E125 ParameterTypes into SEMI E134 ParameterValueTypes.

Table 76 SEMI E134 and SEMI E125.1 Type Description Mapping Table

SEMI E125.1 Type Description complexType	*SEMI E134.1 Parameter Value Element Names*	*XML Schema Type of Parameter Values*
StringType	S	xsd:string
BooleanType	B	xsd:boolean
Base64BinaryType	B64	xsd:base64Binary
ByteType, ShortType, IntType, LongType	I1, I2, I4, I8	xsd:byte, xsd:short, xsd:int, xsd:long
FloatType, DoubleType	F4, F8	xsd:float, xsd:double
DateTimeType	D	xsd:dateTime
AnyURIType	URI	xsd:anyURI
VariableType	Var	dcm: CompositeValueType, See ¶7.2.18.4.16
ArrayType	Arr	dcm: CompositeValueType, See ¶7.2.18.4.16
StructureType	Su	dcm: CompositeValueType, See ¶7.2.18.4.16
EnumeratedType	ES, EI	dcm: CompositeValueType, See ¶7.2.18.4.16

7.2.18.4.6 *NullType* — This element is an extension to the E134 base document. There is no direct corollary in SEMI E134 or SEMI E125 to this type. This type is used to represent a null value as a ParameterValue.

7.2.18.4.7 *F4 & F8* — These elements are the XML Schema representations of the SEMI E134 RealValue class. The following table describes the mapping of these UML class to these XML Schema elements.

Table 77 Translation Table for F4 & F8

Attribute or Role Name	*UML Type*	*XML Element or Attribute (or Reference)*	*XML Name/Type*
realVal	Real	Attribute	Value: xsd:float, xsd:double

7.2.18.4.8 *I1, I2, I4 & I8* — These elements are the XML Schema representations of the SEMI E134 IntegerValue class. The following table describes the mapping of these UML class to these XML Schema elements.

Table 78 Translation Table for I1, I2, I4 & I8

Attribute or Role Name	*UML Type*	*XML Element or Attribute (or Reference)*	*XML Name/Type*
intVal	Integer	Attribute	Value: xsd:byte, xsd:short, xsd:int, xsd:long

7.2.18.4.9 *S* — This element is the XML Schema representations of the SEMI E134 StringValue class. The following table describes the mapping of this UML class to the XML Schema element.

Table 79 Translation Table for S

Attribute or Role Name	*UML Type*	*XML Element or Attribute (or Reference)*	*XML Name/Type*
stringVal	Text	Attribute	Value: xsd:string

7.2.18.4.10 *B* — This element is the XML Schema representations of the SEMI E134 BooleanValue class. The following table describes the mapping of this UML class to the XML Schema element.

Table 80 Translation Table for B

Attribute or Role Name	*UML Type*	*XML Element or Attribute (or Reference)*	*XML Name/Type*
boolVal	Boolean	Attribute	Value: xsd:boolean

7.2.18.4.11 *EI* — This element is the XML Schema representations of the SEMI E134 EnumeratedIntegerValue class. The following table describes the mapping of this UML class to the XML Schema element.

Table 81 Translation Table for EI

Attribute or Role Name	*UML Type*	*XML Element or Attribute (or Reference)*	*XML Name/Type*
intEnumVal	Integer	Attribute	Value: xsd:int

7.2.18.4.12 *ES* — This element is the XML Schema representations of the SEMI E134 EnumeratedStringValue class. The following table describes the mapping of this UML class to the XML Schema element.

Table 82 Translation Table for ES

Attribute or Role Name	*UML Type*	*XML Element or Attribute (or Reference)*	*XML Name/Type*
stringEnumVal	Text	Attribute	Value: xsd:string

7.2.18.4.13 *B64* — This element is the XML Schema representations of the SEMI E134 BinaryValue class. The following table describes the mapping of this UML class to these XML Schema element.

Table 83 Translation Table for B64

Attribute or Role Name	*UML Type*	*XML Element or Attribute (or Reference)*	*XML Name/Type*
binVal	Binary	Attribute	Value: xsd:base64Binary,

7.2.18.4.14 *NoValue* — This element is the XML Schema representation of the SEMI E134 NoValue class. The following table describes the mapping of the UML class to this XML Schema element.

Table 84 Translation Table for NoValue

Attribute or Role Name	*UML Type*	*XML Element or Attribute (or Reference)*	*XML Name/Type*
reasonCode	Enumerated, of type NoValueReasonEnum	Attribute	reasonCode: dcm:NoValueReasonEnumType
description	Text	Attribute	description: xsd:string

7.2.18.4.15 *NoValueReasonEnum* — This elements is the XML Schema representation of the SEMI E134 NoValueReasonEnum class. NoValueReasonEnumType is an xsd:string based enumeration with the following available values.

7.2.18.4.16 *FieldValueType* — This element is the XML Schema container class representation of the SEMI E134 ParmeterValue class. The following table describes the mapping of the UML class to this XML Schema elements. The FieldValueType class references the base classes defined in ¶¶7.2.18.4.7 through 7.2.18.4.17.

Table 85 NoValueReasonEnum Enumerated Values

Enumerated Values
ValueNotAvailable
NoSuchSource
NoSuchParameter

Table 86 Translation Table for FieldValueType

Attribute or Role Name	*UML Type*	*XML Element or Attribute (or Reference)*	*XML Name/Type*
value	RealValue	Element	F4: xsd:float
value	RealValue	Element	F8: xsd:double
value	IntegerValue	Element	I8: xsd:long
value	IntegerValue	Element	I4: xsd:int
value	IntegerValue	Element	I2: xsd:short
value	IntegerValue	Element	I1: xsd:byte
value	StringValue	Element	S: xsd:string
value	SEMI E134.1 Extension	Element	D: xsd:dateTime
value	BooleanValue	Element	B: xsd:boolean
value	SEMI E134.1 Extension	Element	URI: xsd:anyURI
value	EnumeratedIntegerValue	Element	EI: xsd:int
value	EnumeratedStringValue	Element	ES: xsd:string
value	BinaryValue	Element	B64: xsd:base64Binary
value	StructureValue	Element	Su: dcm: CompositeValueType
value	ArrayValue	Element	Arr: dcm: CompositeValueType
value	SEMI E134.1 Extension	Element	Var: dcm: CompositeValueType
SEMI E134.1 Extension	SEMI E134.1 Extension	Element	Null: dcm:NullType
value	NoValue	Element	NoValue: dcm:NoValueType

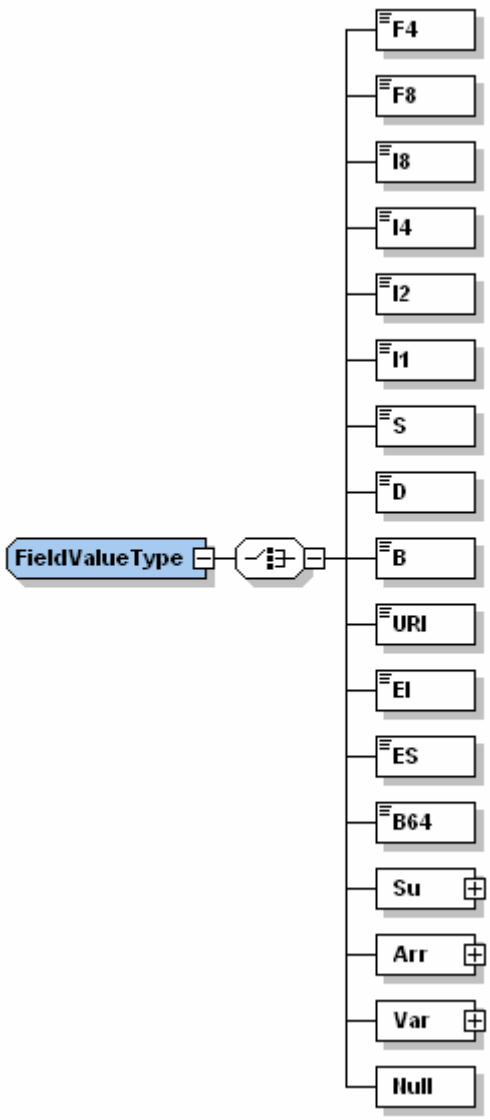

Figure 29
FieldValueType

7.2.18.4.17 *CompositeValueType* — This elements is the XML Schema representation of the SEMI E125.1 VariableTypeType class. The following table describes the mapping of the UML class to this XML Schema elements. The CompositeValueType class references the base classes defined in ¶¶7.2.18.4.7 through 7.2.18.4.17.

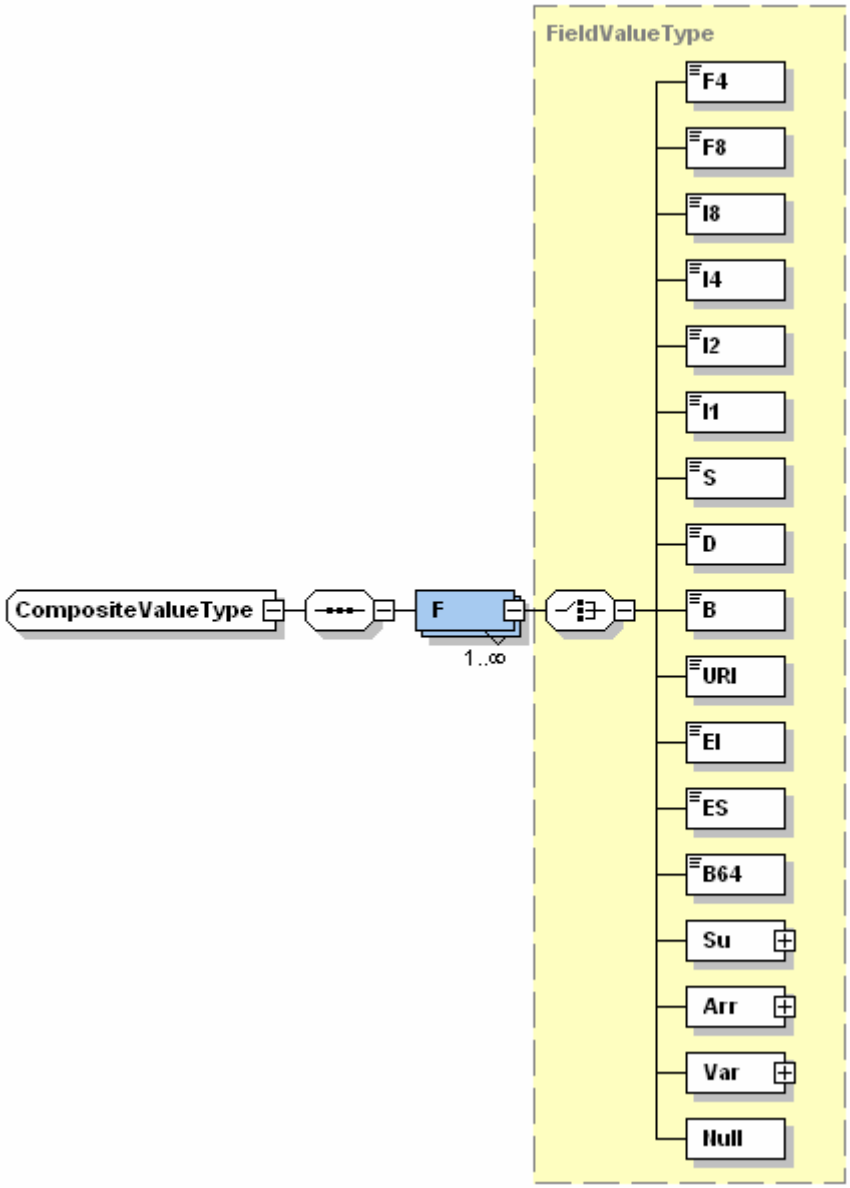

Figure 30
CompositeValueType

7.2.19 *GetObjTypeInstanceIds*

7.2.19.1 *WSDL Operation Binding*

7.2.19.1.1 See the operation binding for "GetObjTypeInstanceIds" in E134-Equipment-Binding.wsdl. Key information is shown for convenience in the following table.

Table 87 GetObjTypeInstanceIds Operation Binding

SOAPAction	urn:semi-org:ws.E134-1.V0305.DCMEqp-binding:GetObjTypeInstanceIds
Input Headers (WSDL Message, Required)	E132HeaderMessage, required
Output Headers (WSDL Message, Required)	E132HeaderMessage, required

7.2.19.2 *WSDL Operation*

7.2.19.2.1 See the portType operation definition for "GetObjTypeInstanceIds" in E134-Equipment-PortType.wsdl. Key information is shown for convenience in the following table.

Table 88 GetObjTypeInstanceIds PortType Operation

Input Message Name	GetObjTypeInstanceIdsRequest
Output Message Name	GetObjTypeInstanceIdsResponse

7.2.19.3 *WSDL Message(s)*

7.2.19.3.1 See the message definitions for "E132HeaderMessage", "GetObjTypeInstanceIdsRequest", and "GetObjTypeInstanceIdsResponse" in E134-Equipment-PortType.wsdl. Key information is shown for convenience in the following table.

Table 89 DataCollectionManagement Binding

Message Name → Schema Element Name	E132HeaderMessage → auth:E132Header GetObjTypeInstanceIdsRequest → dcm:GetObjTypeInstanceIdsRequestType GetObjTypeInstanceIdsResponse → dcm:GetObjTypeInstanceIdsResponseType

7.2.19.4 *XML Schema Types*

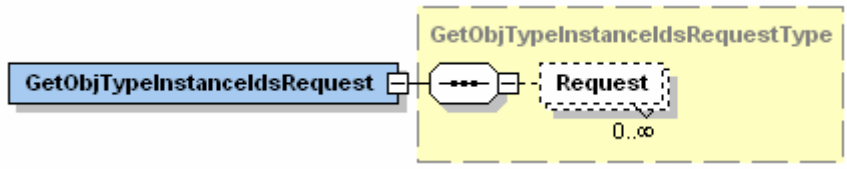

Figure 31
GetObjTypeInstanceIdsRequest

7.2.19.4.1 *GetObjTypeInstanceIdsRequest* — This global element acts as a top-level container entity representing the SOAP body of a GetObjTypeInstanceIds request. As defined in SEMI E134, this operation has one input argument: the Parameters that are being requested. This argument is modeled as an element named "Request" of complexType "ObjTypeRequestType".

Table 90 Translation Table for Input Arguments for the GetObjTypeInstanceIds Operation

Argument Name	*Format*	*XML Element or Attribute*	*XML Name/Type*
request	List of elements of type ObjTypeRequest	Element	Request: dcm:ObjTypeRequestType

7.2.19.4.2 *ObjTypeRequestType* — This global complexType is the XML Schema representation of the SEMI E134 ObjTypeRequest class. The following table describes the mapping of the UML class to this XML Schema complexType.

Table 91 Translation Table for ObjTypeRequestType

Attribute or Role Name	*UML Type*	*XML Element or Attribute (or Reference)*	*XML Name/Type*
sourceId	Text	Attribute	sourceId: xsd:string
objTypeId	Text	Attribute	objTypeId: xsd:string

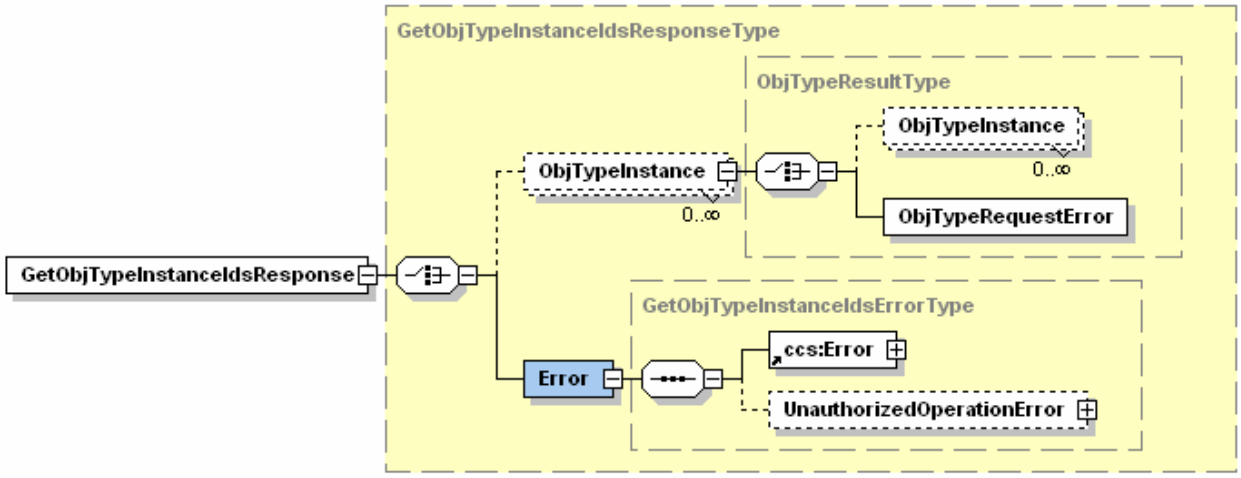

Figure 32
GetObjTypeInstanceIdsResponse

7.2.19.4.3 *GetObjTypeInstanceIdsResponse* — This global element acts as a top-level container entity representing the SOAP body of the response to an SEMI E134 GetObjTypeInstanceIds request. It contains either an element named “objTypeInstances” of complexType “ObjTypeResultType” or an element named “Error” of complexType “dcm:GetObjTypeInstanceIdsErrorType”.

Table 92 Translation Table for Output Arguments for the GetObjTypeInstanceIds Operation

Argument Name	*Format*	*XML Element or Attribute*	*XML Name/Type*
objTypeInstances	List of elements of type ParameterValue	Element	ObjTypeInstances: dcm:ObjTypeResultType
error	N/A	Element	ErrorType: dcm:GetObjTypeInstanceIdsErrorType

7.2.19.4.4 *ObjTypeResultType* — This global complexType is the XML Schema representation of the SEMI E134 ObjTypeResult class. The following table describes the mapping of the UML class to this XML Schema complexType.

Table 93 Translation Table for ObjTypeResultType

Attribute or Role Name	*UML Type*	*XML Element or Attribute (or Reference)*	*XML Name/Type*
objTypeInstances	List of zero or more elements of type ObjTypeInstance	Element	ObjTypeInstances: dcm:ObjTypeInstanceArrayType
objTypeRequestError	Zero or one element of type ObjTypeRequestError	Element	ObjTypeRequestError: dcm:ObjTypeRequestErrorType
sourceId	Text	Attribute	sourceId: xsd:string
objTypeId	Text	Attribute	objTypeId: xsd:string

7.2.19.4.5 *ObjTypeInstanceType* — This global complexType is the XML Schema representation of the SEMI E134 ObjTypeInstance class. The following table describes the mapping of the UML class to this XML Schema complexType

Table 94 Translation Table for ObjTypeInstanceType

Attribute or Role Name	*UML Type*	*XML Element or Attribute (or Reference)*	*XML Name/Type*
instanceId	Text	Attribute	instanceId: xsd:string

7.2.19.4.6 *ObjTypeRequestErrorType* — This is a container class representing the errors that can be returned from the ObjTypeRequest message. ObjTypeRequestErrorType has the ability to specify specific errors as described in the following table. The following table describes the mapping of the UML class to this XML Schema complexType

Table 95 Translation Table for ObjTypeRequestErrorType

Attribute or Role Name	*UML Type*	*XML Element or Attribute (or Reference)*	*XML Name/Type*
invalidSource	Boolean	Attribute	invalidSource: xsd:boolean
invalidObjType	Boolean	Attribute	invalidObjType: xsd:boolean
sourceDoesNotProduceObjType	Boolean	Attribute	sourceDoesNotProduceObjType: xsd:boolean

7.2.20 *GetCurrentPerformanceStatus*

7.2.20.1 *WSDL Operation Binding*

7.2.20.1.1 See the operation binding for "GetCurrentPerformanceStatus" in E134-Equipment-Binding.wsdl. Key information is shown for convenience in the following table.

Table 96 GetCurrentPerformanceStatus Operation Binding

SOAPAction	urn:semi-org:ws.E134-1.V0305.DCMEqp-binding:GetCurrentPerformanceStatus
Input Headers (WSDL Message, Required)	E132HeaderMessage, required
Output Headers (WSDL Message, Required)	E132HeaderMessage, required

7.2.20.2 *WSDL Operation*

7.2.20.2.1 See the portType operation definition for "GetCurrentPerformanceStatus" in E134-Equipment-PortType.wsdl. Key information is shown for convenience in the following table.

Table 97 GetCurrentPerformanceStatus PortType Operation

Input Message Name	GetCurrentPerformanceStatusRequest
Output Message Name	GetCurrentPerformanceStatusResponse

7.2.20.3 *WSDL Message(s)*

7.2.20.3.1 See the message definitions for "E132HeaderMessage", "GetCurrentPerformanceStatusRequest", and "GetCurrentPerformanceStatusResponse" in E134-Equipment-PortType.wsdl. Key information is shown for convenience in the following table.

Table 98 DataCollectionManagement Binding

Message Name → Schema Element Name	E132HeaderMessage → auth:E132Header GetCurrentPerformanceStatusRequest → dcm:GetCurrentPerformanceStatusRequestType GetCurrentPerformanceStatusResponse → dcm:GetCurrentPerformanceStatusResponseType

7.2.20.4 *XML Schema Types*

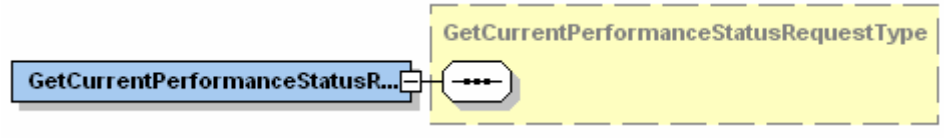

Figure 33
GetCurrentPerformanceStatusRequest

7.2.20.4.1 *GetCurrentPerformanceStatusRequest* — This global element acts as a top-level container entity representing the SOAP body of a GetCurrentPerformanceStatus request. As defined in SEMI E134, this operation has no input arguments.

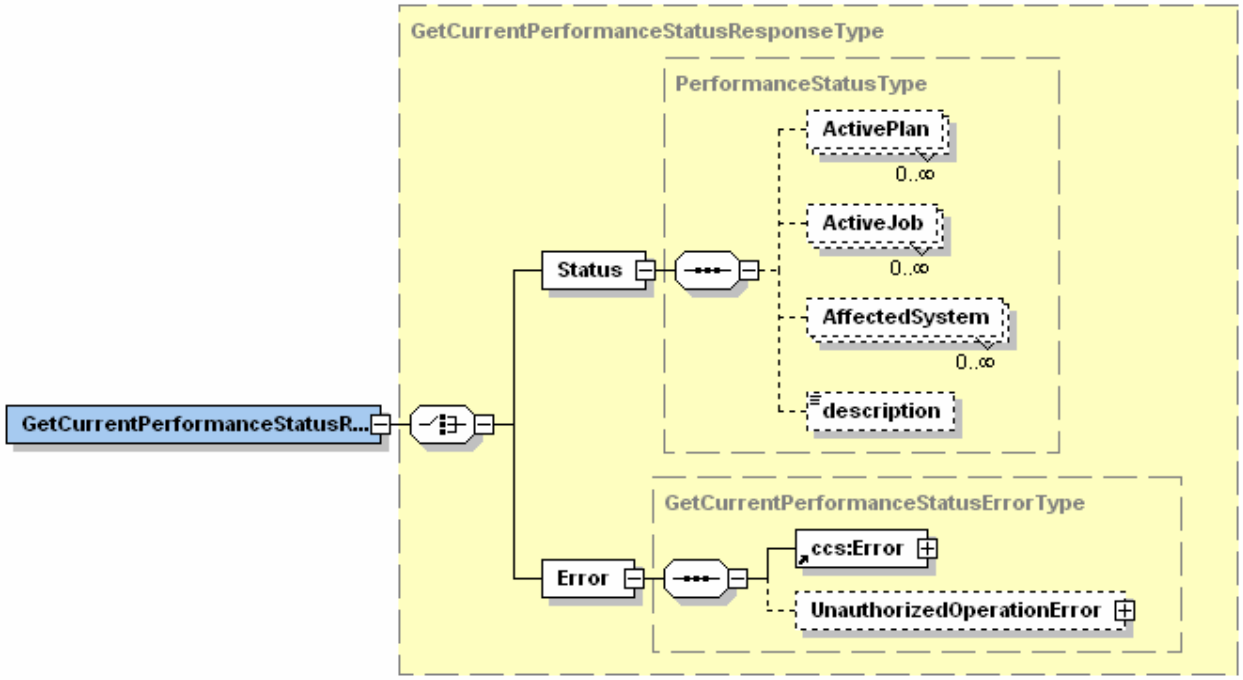

Figure 34
GetCurrentPerformanceStatusResponse

7.2.20.4.2 *GetCurrentPerformanceStatusResponse* — This global element acts as a top-level container entity representing the SOAP body of the response to an SEMI E134 GetCurrentPerformanceStatus request. It contains either an element named “objTypeInstances” of complexType “ObjTypeResultArrayType.

Table 99 Translation Table for Output Arguments for the GetCurrentPerformanceStatus Operation

Argument Name	*Format*	*XML Element or Attribute*	*XML Name/Type*
status	One and only one element of type PerformanceStatus	Element	Status: dcm:PerformanceStatusType
error	N/A	Element	ErrorType: ccs:Error
error	Structured data, of type UnauthorizedOperation	Element	UnauthorizedOperationError: eca:UnauthorizedOperationType, see ¶7.2.11.4.12

7.2.20.4.3 *PerformanceStatusType* — This global complexType is the XML Schema representation of the SEMI E134 PerformanceStatus class. The following table describes the mapping of the UML class to this XML Schema complexType.

Table 100 Translation Table for PerformanceStatusType

Attribute or Role Name	*UML Type*	*XML Element or Attribute (or Reference)*	*XML Name/Type*
ActivePlans	Unordered list of elements of type DCPActivated	Element	ActivePlans: dcm:DCPActivatedArrayType as described in ¶7.2.14.4.3
ActiveJobs	Unordered list of elements of type ActiveJob	Element	ActiveJobs: dcm:ActiveJobArrayType
AffectedSystems	Unordered list of zero or more elements of type AffectedSystem	Element	AffectedSystems: dcm:AffectedSystemArrayType
timeLastEvaluated	Text	Attribute	timeLastEvaluated: xsd:dateTime
isBelowThreshold	Boolean	Attribute	isBelowThreshold: xsd:boolean
description	Text	Element	description: xsd:string

7.2.20.4.4 *ActiveJobType* — This global complexType is the XML Schema representation of the SEMI E134 ActiveJob class. The following table describes the mapping of the UML class to this XML Schema complexType.

Table 101 Translation Table for ActiveJobType

Attribute or Role Name	*UML Type*	*XML Element or Attribute (or Reference)*	*XML Name/Type*
jobId	Text	Attribute	jobId: xsd:string
jobType	Text	Attribute	jobType: dcm:JobTypeEnumType

7.2.20.4.5 *JobTypeEnumType* — JobTypeEnumType is a string restricted to the following supported values.

Table 102 Supported Job Type Table for JobTypeEnumType

Supported JobTypes
urn:semi-org:E94:ControlJob
urn:semi-org:E40:ProcessJob
urn:semi-org:E30:ppSelect
urn:semi-org:E30:rcpSelect

7.2.20.4.6 *AffectedSystemType* — This global complexType is the XML Schema representation of the SEMI E134 AffectedSystem class. The following table describes the mapping of the UML class to this XML Schema complexType.

Table 103 Translation Table for AffectedSystemType

Attribute or Role Name	*UML Type*	*XML Element or Attribute (or Reference)*	*XML Name/Type*
equipmentComponent	Text	Attribute	equipmentComponent: xsd:string
description	Text	Attribute	description: xsd:string

7.3 *DCPConsumer*

7.3.1 *XML Schema and WSDL Files*

7.3.1.1 This section is provisional, pending the approval of SEMI E120.1, SEMI E125.1, and SEMI E132.1.

7.3.1.2 The XML schema and WSDL defined by this specification for the DCPConsumer interface is contained in the following documents:

Table 104 XML Schema

File Name	E134-1-V0305-Schema.xsd
Target Namespace	urn:semi-org:xsd.E134-1.V0305.DCM
Imported/Referenced Namespaces	urn:semi-org:xsd.E120-1.V0704.CommonEquipmentModel http://www.w3.org/2001/XMLSchema
Description	This file defines all of the SEMI E134 data types and elements used by both the DataCollectionManager and DCPConsumer interfaces.

Table 105 DCPConsumer PortType Definitions

File Name	E134-1-V0305-Client-portType.wsdl
Target Namespace	urn:semi-org:ws.E134-1.V0305.DCMClient-portType
Imported/Referenced Namespaces	urn:semi-org:xsd.E132-1.V0305.auth urn:semi-org:xsd.E134-1.V0305.DCM http://schemas.xmlsoap.org/wsdl/ http://www.w3.org/2001/XMLSchema
Description	This file defines all of the input/output messages and operations for the DCPConsumer interface, based on the data types defined in the SEMI E134 XML Schema.

Table 106 DCPConsumer Binding Definitions

File Name	E1341-V0305--Client-binding.wsdl
Target Namespace	urn:semi-org:ws.E134-1.V0305.DCMClient-binding
Imported/Referenced Namespaces	urn:semi-org:ws.E134-1.V0305.DCMClient-portType http://schemas.xmlsoap.org/wsdl/soap/ http://schemas.xmlsoap.org/wsdl/
Description	This file binds the abstract portType definition to HTTP and SOAP, specifying required SOAP and HTTP headers for each operation and the XML encoding style.

7.3.1.3 These documents are a part of this specification and should accompany this document. The contents of these documents constitute the core part of this specification.

7.3.1.4 E134-1-V0305-Schema.xsd imports types from the SEMI E132.1 Equipment Client Authentication and Authorization XML Schema namespace, specifically the E132Header and E132HashHeader types.

7.3.1.5 E134-1-V0305-Schema.xsd imports types from the SEMI 3570.1 Specification for XML Semiconductor Common Components XML Schema namespace, specifically the ErrorType type.

7.3.2 *WSDL Port Type Overview*

7.3.2.1 The DCPConsumer WSDL portType definition organizes the SEMI E134.1 XML Schema data types into a collection of named operations with inputs and outputs that correspond to the UML operations that are defined for the DCPConsumer interface in SEMI E134, ¶9.1. The WSDL portType itself is named after the SEMI E134 interface, and each WSDL portType operation is named after the corresponding UML operation defined for the interface. Each WSDL operation consists of two WSDL message definitions corresponding to the input and output (request and response) for the UML operation defined in SEMI E134. The WSDL message and operation definitions are described in ¶¶7.3.5 through 7.3.9.

7.3.3 *WSDL Binding Overview*

7.3.3.1 The WSDL DCPConsumer binding definition specifies a message and transport protocol to use for a given portType (SOAP and HTTP for SEMI E134.1), the interface style used (document style for SEMI E134.1). For each WSDL portType operation, the binding defines any SOAP headers used and whether or not they are required, the HTTP SOAPAction header value to use for that operation, and the XML encoding to use (literal). The binding definitions for each portType operation are described in ¶¶7.3.5 through 7.3.9.

7.3.4 *WSDL Service Overview*

7.3.4.1 This specification does not provide a WSDL service definition. Client suppliers shall provide a WSDL service definition file with the client that supports E134.1. This service description shall conform to the WS-I Basic Profile 1.0a, ¶7.1, for which the HTTPS transport with mutual authentication is required.

7.3.4.2 *Application of SEMI E132 Sessions*

7.3.4.3 Operations defined by the SEMI E134 DCPConsumer interface shall be exchanged via HTTP and shall not be exchanged via HTTPS. The equipment shall include a hash of the client's SEMI E132 session identifier in the SOAP envelope header provided with the operation using the E132HashHeader element, as specified by SEMI E132.1

7.3.4.4 The equipment shall only send SEMI E134 notifications to instances of the DCPConsumer interface if the corresponding SEMI E132 Session(s) for that client are in the "Established" state. The equipment shall use the URL provided with the HTTPEndpoint information included in the SEMI E132 EstablishSession request to send all SEMI E134 notifications to the DCPConsumer interface.

7.3.5 *NewData*

7.3.5.1 *WSDL Operation Binding*

7.3.5.1.1 See the operation binding for "NewData" in E134-Client-Binding.wsdl. Key information is shown for convenience in the following table.

Table 107 NewData Operation Binding

SOAPAction	urn:semi-org:ws.E134-1.V0305.DCPConsumer-binding:NewData
Input Headers (WSDL Message, Required)	E132HashHeaderMessage, required

7.3.5.2 *WSDL Operation*

7.3.5.2.1 See the portType operation definition for "NewData" in E134-Client-PortType.wsdl. Key information is shown for convenience in the following table.

Table 108 NewData PortType Operation

Input Message Name	NewDataNotification

7.3.5.3 *WSDL Message(s)*

7.3.5.3.1 See the message definitions for "E132HashHeaderMessage" and "NewDataNotification" in E134-Client-PortType.wsdl. Key information is shown for convenience in the following table.

Table 109 DCPConsumer Binding

Message Name → Schema Element Name	E132HashHeaderMessage → auth:E132HashHeader NewDataNotification → dcm:NewDataNotificationType

7.3.5.4 *XML Schema Types*

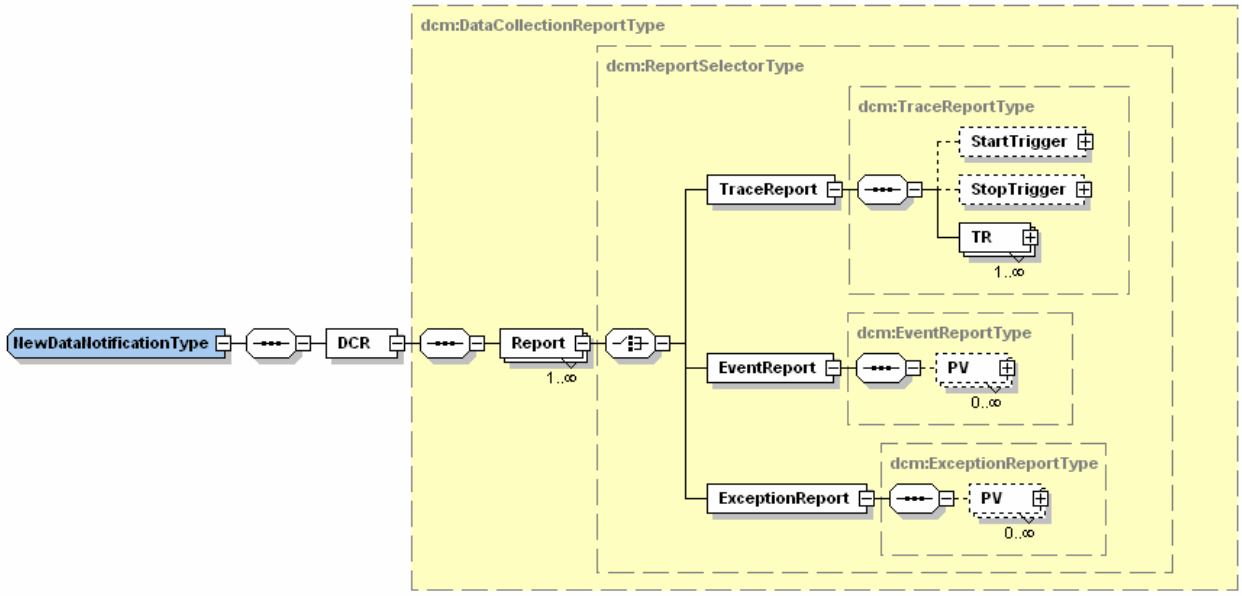

Figure 35
NewDataNotification

7.3.5.4.1 *NewDataNotification* — This global element acts as a top-level container entity representing the SOAP body of a NewData Notification. As defined in SEMI E134, this operation has one output argument: the Data Collection Report, DCR. This argument is modeled as an element named "DCR" of complexType "dcm:DataCollectionReportType".

Table 110 Translation Table for Input Arguments for the NewDataNotification Operation

Argument Name	*Format*	*XML Element or Attribute*	*XML Name/Type*
dataCollectionReport	Structured data, of type DataCollectionReport	Element	DCR: dcm:DataCollectionReportType

7.3.5.4.2 *DataCollectionReportType* — This global complexType is the XML Schema representation of the SEMI E134 DataCollectionReport class. The following table describes the mapping of the UML class to this XML Schema complexType

Table 111 Translation Table for DataCollectionReportType

Attribute or Role Name	*UML Type*	*XML Element or Attribute (or Reference)*	*XML Name/Type*
reports	1 or more elements of any type derived from Report	Element	Report: dcm:ReportSelectorType
planId	Text	Attribute	planId: xsd:string
bufferStartTime	Text	Attribute	bufferStartTime: xsd:dateTime
bufferEndTime	Text	Attribute	bufferEndTime: xsd:dateTime
reportTime	Text	Attribute	reportTime: xsd:dateTime

7.3.5.4.3 *ReportSelectorType* — This global complexType is the XML Schema representation of the SEMI E134 Report class. The following table describes the mapping of the UML class to this XML Schema complexType.

Table 112 Translation Table for ReportSelectorType

Attribute or Role Name	*UML Type*	*XML Element or Attribute (or Reference)*	*XML Name/Type*
TraceReport	Structured data, of type TraceReport	Element	TraceReport: dcm:TraceReportType
EventReport	Structured data, of type EventReport	Element	EventReport: dcm:EventReportType
EventReport	Structured data, of type ExceptionReport	Element	ExceptionReport: dcm:ExceptionReportType

7.3.5.4.4 *EventReportType* — This global complexType is the XML Schema representation of the SEMI E134 DataCollectionReport class. The following table describes the mapping of the UML class to this XML Schema complexType.

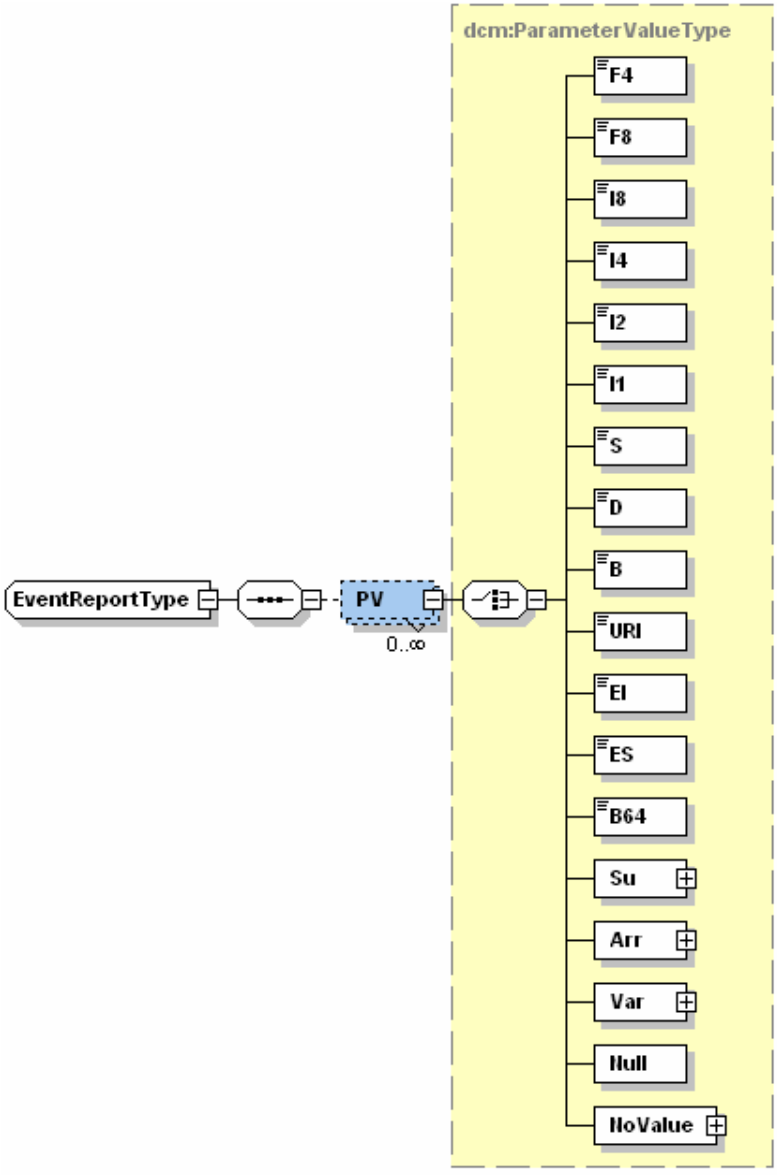

Figure 36
EventReportType

Table 113 Translation Table for EventReportType

Attribute or Role Name	*UML Type*	*XML Element or Attribute (or Reference)*	*XML Name/Type*
parameterValues	Ordered list of zero or more elements of type ParameterValue	Element	PV: dcm:ParameterValueType
sourceId	Text	Attribute	sourceId: xsd:string
eventId	Text	Attribute	eventId: xsd:string
eventTime	Text	Attribute	eventTime: xsd:dateTime

7.3.5.4.5 *ExceptionReportType* — This global complexType is the XML Schema representation of the SEMI E134 ExceptionReport class. The following table describes the mapping of the UML class to this XML Schema complexType.

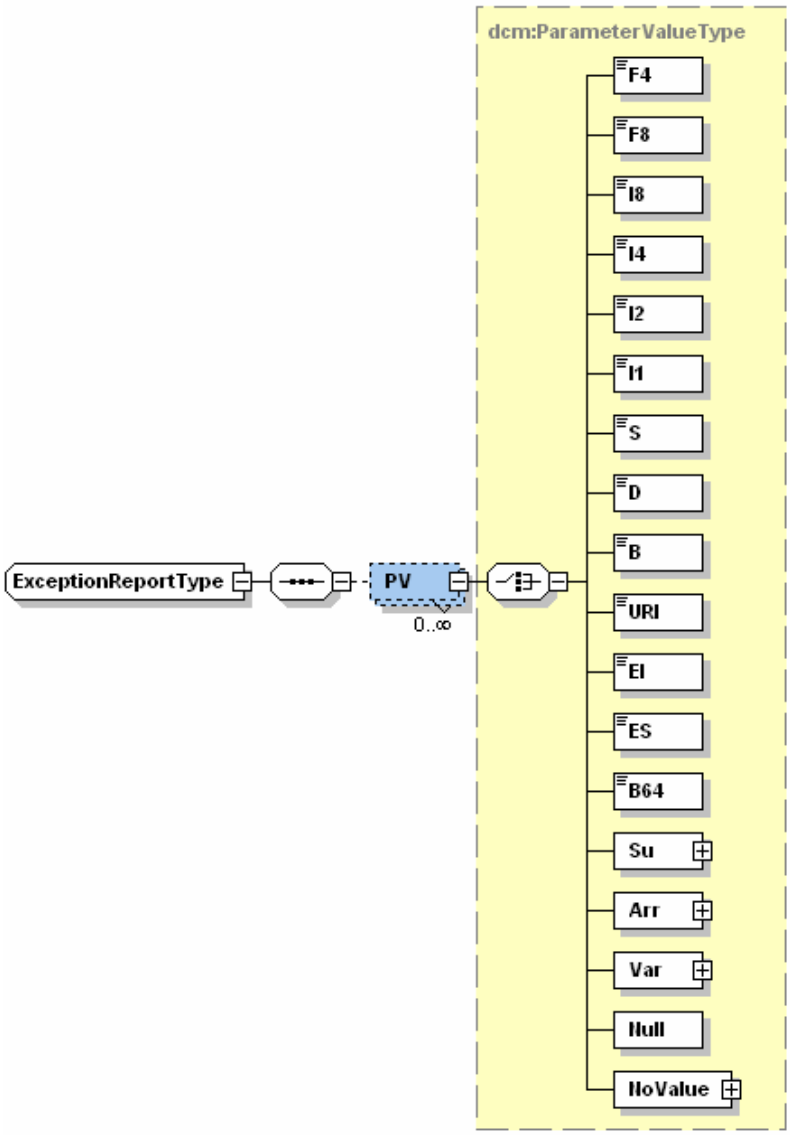

Figure 37
ExceptionReportType

Table 114 Translation Table for ExceptionReportType

Attribute or Role Name	*UML Type*	*XML Element or Attribute (or Reference)*	*XML Name/Type*
parameterValues	Ordered list of zero or more elements of type ParameterValue	Element	PV: dcm:ParameterValueType
sourceId	Text	Attribute	sourceId: xsd:string
exceptionId	Text	Attribute	exceptionId: xsd:string
exceptionTime	Text	Attribute	exceptionTime: xsd:dateTime
severity	Text	Attribute	severity: xsd:string
state	Text	Attribute	state: xsd:string

7.3.5.4.6 *TraceReportType* — This global complexType is the XML Schema representation of the SEMI E134 TraceReport class. The following table describes the mapping of the UML class to this XML Schema complexType.

Table 115 Translation Table for TraceReportType

Attribute or Role Name	*UML Type*	*XML Element or Attribute (or Reference)*	*XML Name/Type*
traceResults	Ordered list of one or more elements of type CollectedData, appearing in the order of each result's start time	Element	TR: dcm:CollectedDataType
startTrigger	Zero or one element of any type derived from Trigger	Element	StartTrigger: dcm:TriggerSelectorType (see Figure 9 and ¶¶7.2.11.4.7and 7.2.11.4.8
stopTrigger	Zero or one element of any type derived from Trigger	Element	StopTrigger: dcm:TriggerSelectorType (see Figure 9 and ¶¶7.2.11.4.7and 7.2.11.4.8
traceId	Integer	Attribute	sourceId: xsd:int
startTriggerTime	Text	Attribute	startTriggerTime: xsd:dateTime
stopTriggerTime	Text	Attribute	stopTriggerTime: xsd:dateTime
reportTime	Text	Attribute	reportTime: xsd:dateTime

7.3.5.4.7 *CollectedDataType* — This global complexType is the XML Schema representation of the SEMI E134 CollectedData class. The following table describes the mapping of the UML class to this XML Schema complexType.

7.3.5.4.8 *CollectedDataType* — This global complexType is the XML Schema representation of the SEMI E134 CollectedData class. The following table describes the mapping of the UML class to this XML Schema complexType.

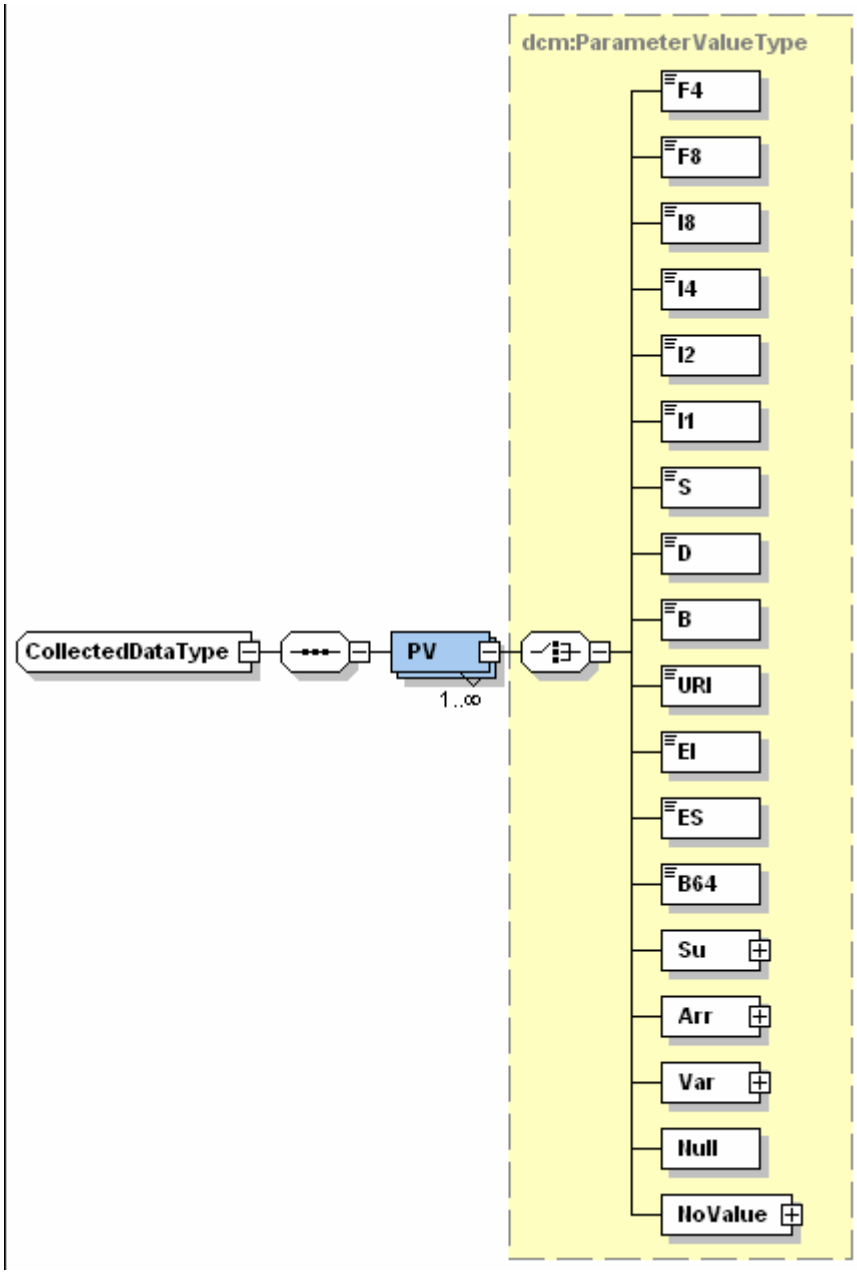

Figure 38
CollectedDataType

Table 116 Translation Table for CollectedDataType

Attribute or Role Name	*UML Type*	*XML Element or Attribute (or Reference)*	*XML Name/Type*
parameterValues	Ordered list of zero or more elements of type ParameterValue	Element	PV: dcm:ParameterValueType
collectionTime	Text	Attribute	collectionTime: xsd:dateTime

7.3.6 *PerformanceWarning*

7.3.6.1 *WSDL Operation Binding*

7.3.6.1.1 See the operation binding for “PerformanceWarning” in E134-Client-Binding.wsdl. Key information is shown for convenience in the following table.

Table 117 PerformanceWarning Operation Binding

SOAPAction	urn:semi-org:ws.E134-1.V0305.DCPConsumer-binding:PerformanceWarning
Input Headers (WSDL Message, Required)	E132HashHeaderMessage, required

7.3.6.2 *WSDL Operation*

7.3.6.2.1 See the portType operation definition for “PerformanceWarning” in E134-Client-PortType.wsdl. Key information is shown for convenience in the following table.

Table 118 PerformanceWarning PortType Operation

Input Message Name	PerformanceWarningNotification

7.3.6.3 *WSDL Message(s)*

7.3.6.3.1 See the message definitions for “E132HashHeaderMessage” and “PerformanceWarningNotification” in E134-Client-PortType.wsdl. Key information is shown for convenience in the following table.

Table 119 DCPConsumer Binding

Message Name → Schema Element Name	E132HashHeaderMessage → auth:E132HashHeader PerformanceWarningNotification → dcm:PerformanceWarningNotificationType

7.3.6.4 *XML Schema Types*

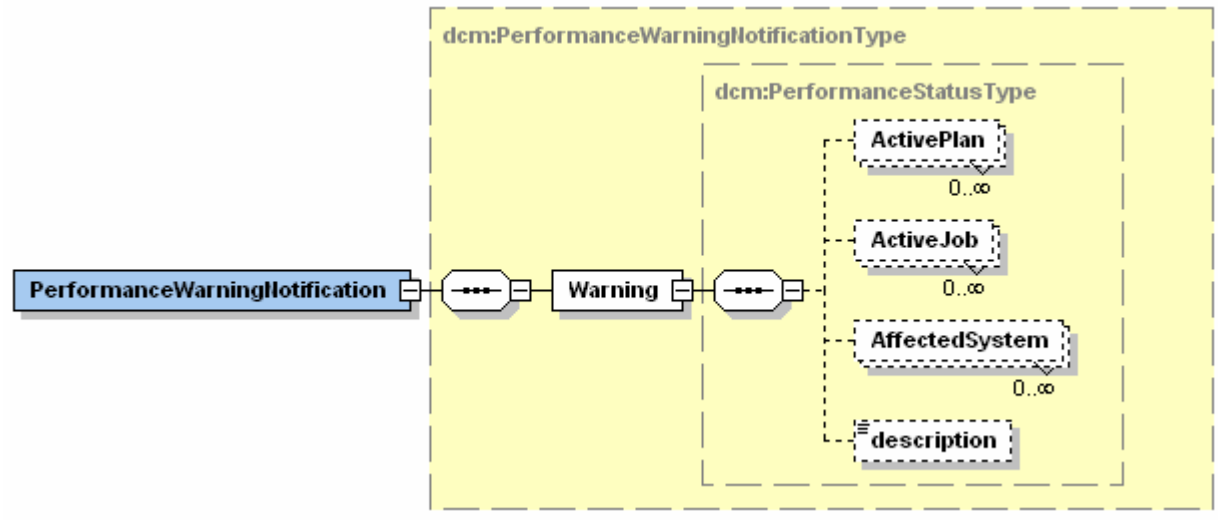

Figure 39
PerformanceWarningNotification

7.3.6.4.1 *PerformanceWarningNotification* — This global element acts as a top-level container entity representing the SOAP body of a PerformanceWarning notification. As defined in SEMI E134, this operation has one output argument: the warning. This argument is modeled as an attributes named “Warning” of complexType “PerformanceStatusType”.

Table 120 Translation Table for Input Arguments for the PerformanceWarning Operation

Argument Name	*Format*	*XML Element or Attribute*	*XML Name/Type*
warning	Structured data, of type PerformanceStatus	Element	Warning : dcm: PerformanceStatusType as described in ¶7.2.20.4.3

7.3.7 *PerformanceRestored*

7.3.7.1 *WSDL Operation Binding*

7.3.7.1.1 See the operation binding for "PerformanceRestored" in E134-Client-Binding.wsdl. Key information is shown for convenience in the following table.

Table 121 PerformanceRestored Operation Binding

SOAPAction	urn:semi-org:ws.E134-1.V0305.DCPConsumer-binding:PerformanceRestored
Input Headers (WSDL Message, Required)	E132HashHeaderMessage, required

7.3.7.2 *WSDL Operation*

7.3.7.2.1 See the portType operation definition for "PerformanceRestored" in E134-Client-PortType.wsdl. Key information is shown for convenience in the following table.

Table 122 PerformanceRestored PortType Operation

Input Message Name	PerformanceRestoredNotification

7.3.7.3 *WSDL Message(s)*

7.3.7.3.1 See the message definitions for "E132HashHeaderMessage" and "PerformanceRestoredNotification" in E134-Client-PortType.wsdl. Key information is shown for convenience in the following table.

Table 123 DCPConsumer Binding

Message Name → Schema Element Name	E132HashHeaderMessage → auth:E132HashHeader PerformanceRestoredNotification → dcm:PerformanceRestoredNotificationType

7.3.7.4 *XML Schema Types*

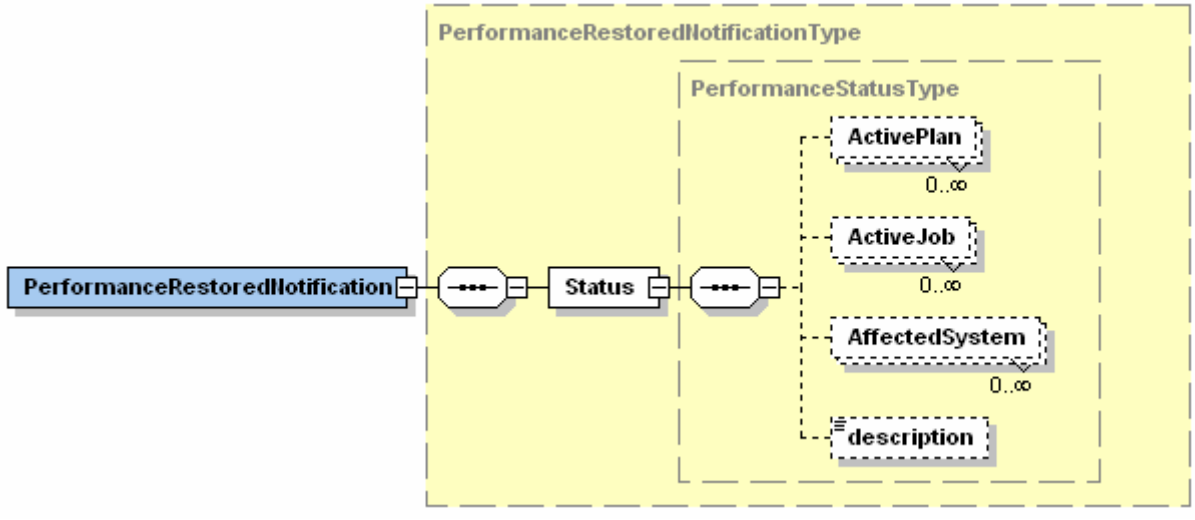

Figure 40
PerformanceRestoredNotification

7.3.7.4.1 *PerformanceRestoredNotification* — This global element acts as a top-level container entity representing the SOAP body of a PerformanceRestored notification. As defined in SEMI E134, this operation has one output argument: the Status. This argument is modeled as an attributes named "Status" of complexType "PerformanceStatusType".

Table 124 Translation Table for Input Arguments for the PerformanceRestored Operation

Argument Name	*Format*	*XML Element or Attribute*	*XML Name/Type*
status	Structured data, of type PerformanceStatus	Element	Status: dcm: PerformanceStatusType as described in ¶7.2.20.4.3

7.3.8 *DCPDeactivation*

7.3.8.1 *WSDL Operation Binding*

7.3.8.1.1 See the operation binding for "DCPDeactivation" in E134-Client-Binding.wsdl. Key information is shown for convenience in the following table.

Table 125 DCPDeactivation Operation Binding

SOAPAction	urn:semi-org:ws.E134-1.V0305.DCPConsumer-binding:DCPDeactivation
Input Headers (WSDL Message, Required)	E132HashHeaderMessage, required

7.3.8.2 *WSDL Operation*

7.3.8.2.1 See the portType operation definition for "DCPDeactivation" in E134-Client-PortType.wsdl. Key information is shown for convenience in the following table.

Table 126 DCPDeactivation PortType Operation

Input Message Name	DCPDeactivationNotification

7.3.8.3 *WSDL Message(s)*

7.3.8.3.1 See the message definitions for "E132HashHeaderMessage" and "DCPDeactivationNotification" in E134-Client-PortType.wsdl. Key information is shown for convenience in the following table.

Table 127 DCPConsumer Binding

Message Name → Schema Element Name	E132HashHeaderMessage → auth:E132HashHeader DCPDeactivationNotification → dcm:DCPDeactivationNotificationType

7.3.8.4 *XML Schema Types*

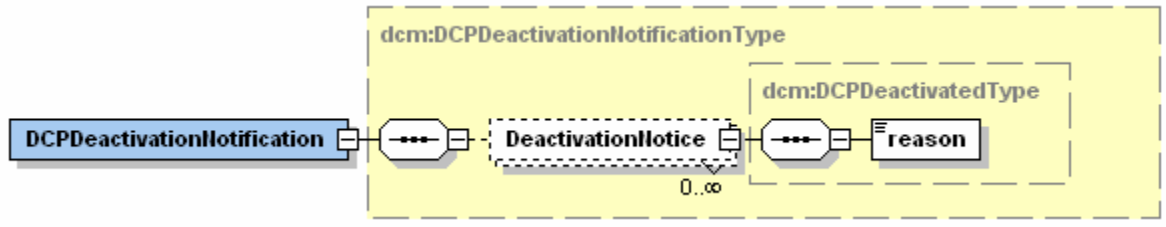

Figure 41
DCPDeactivationNotification

7.3.8.4.1 *DCPDeactivationNotification* — This global element acts as a top-level container entity representing the SOAP body of a DCPDeactivation notification. As defined in SEMI E134, this operation has one output argument: the Deactivation Notice. This argument is modeled as an attributes named "DeactivationNotice" of complexType "DCPDeactivatedType".

Table 128 Translation Table for Input Arguments for the DCPDeactivation Operation

Argument Name	*Format*	*XML Element or Attribute*	*XML Name/Type*
deactivationNotice	List of one or more structured data elements, each of type DCPDeactivated	Element	DeactivationNotice: dcm:DCPDeactivatedType as described in ¶7.2.16.4.3

7.3.9 *DCPHibernation*

7.3.9.1 *WSDL Operation Binding*

7.3.9.1.1 See the operation binding for "DCPHibernation" in E134-Client-Binding.wsdl. Key information is shown for convenience in the following table.

Table 129 DCPHibernation Operation Binding

SOAPAction	urn:semi-org:ws.E134-1.V0305.DCPConsumer-binding:DCPHibernation
Input Headers (WSDL Message, Required)	E132HashHeaderMessage, required

7.3.9.2 *WSDL Operation*

7.3.9.2.1 See the portType operation definition for "DCPHibernation" in E134-Client-PortType.wsdl. Key information is shown for convenience in the following table.

Table 130 DCPHibernation PortType Operation

Input Message Name	DCPHibernationNotification

7.3.9.3 *WSDL Message(s)*

7.3.9.3.1 See the message definitions for "E132HashHeaderMessage" and "DCPHibernationNotification" in E134-Client-PortType.wsdl. Key information is shown for convenience in the following table.

Table 131 DCPConsumer Binding

Message Name → Schema Element Name	E132HashHeaderMessage → auth:E132HashHeader DCPHibernationNotification → dcm:DCPHibernationNotificationType

7.3.9.4 *XML Schema Types*

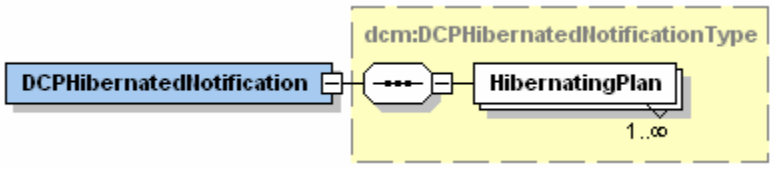

Figure 42
DCPHibernationNotification

7.3.9.4.1 *DCPHibernationNotification* — This global element acts as a top-level container entity representing the SOAP body of a DCPHibernation notification. As defined in SEMI E134, this operation has one output argument: the HibernatingPlans. This argument is modeled as a attributes named "HibernatingPlan" of complexType "DCPHibernatedType".

Table 132 Translation Table for Input Arguments for the DCPHibernation Operation

Argument Name	*Format*	*XML Element or Attribute*	*XML Name/Type*
hibernatingPlans	List of one or more structured data elements, each of type DCPHibernated	Element	HibernatingPlan: dcm:DCPHibernatedType

7.3.9.4.2 *DCPHibernatedType* — This global complexType is the XML Schema representation of the SEMI E134 DCPHibernated class. The following table describes the mapping of the UML class to this XML Schema complexType

Table 133 Translation Table for DCPHibernatedType

Attribute or Role Name	*UML Type*	*XML Element or Attribute (or Reference)*	*XML Name/Type*
planId	Text	Attribute	planId: xsd:string
timeHibernated	Text	Attribute	timeHibernated: xsd:dateTime

SEMI E138-0305
XML SEMICONDUCTOR COMMON COMPONENTS

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

1 Purpose

1.1 This document is a central location for the definition of common XML elements that are not specific to a single standard, but common across multiple standards.

2 Scope

2.1 This document currently defines the representation of Error. This common component is used in association with other XML documents or schemas generated as part of a SEMI standard interface specification for communication between software entities.

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 The purpose of this document is to define basic components and to allow for future additions as common components are discovered and required. Only one component (Error) is being defined at this time.

4 Referenced Standards

4.1 *SEMI Standards*

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

SEMI E87 — Specification for Carrier Management (CMS)

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

SEMI E125 — Specification for Equipment Self Description (EqSD)

SEMI E132 — Specification for Equipment Client Authentication and Authorization

4.2 *Other References*

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

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

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

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

5 Terminology

5.1 *Abbreviations and Acronyms*

5.1.1 *W3C* — World Wide Web Consortium

5.1.2 *WWW* — World Wide Web

5.1.3 *XML* — Extensible Markup Language

6 Resources, Organizations, and Standards

6.1 Important resources and organizations used in the generation and definition of this document are found in:

6.1.1 *XML Schema Part 2: Datatypes* — W3C Recommendation

7 XML Type Definitions

7.1 *XML Schema* — This specification includes an XML schema file, "CommonComponents-V0305-Schema.xsd", and should be provided with this document. That XML schema file provides the normative definitions of the types described here.

7.2 *Data Type for Error Handling* — Error handling in most cases even with standard interface specifications is implementation-specific. It is not practical for one specification to define all possible error conditions, error codes, descriptions and supplementary information that is needed to properly recover from the error. Regardless, a common approach to communicating error conditions can simplify error management for standards implementers and application developers.

7.3 *Error Data Structure* — The following data structure defines a common error structure that can be extended with additional implementation-specific information that can be used by other standards or suppliers.

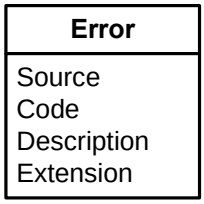

Figure 1
Error Structure

7.3.1 *Source* — The "Source" attribute represents the document from where the attribute code value is defined. For error codes defined by SEMI standards, the content of Source is the urn representation of the standard number. For error codes defined by a supplier, it is the urn representation of the supplier's name. (i.e. For SEMI E132 errors, the content of "Source" is: urn:semi-org:E132; for supplier ACME errors the content is: urn:ACME-com) Additional information may be included by the supplier by adding such information after the ":" colon symbol and separating the fields with a "." period symbol.

Table 1 Example of Error Sources

#	*Source*	*Comment*
1	urn:semi-org:E005	Standard Error Codes for Equipment - SEMI E5.
2	urn:semi-org:E087	Standard Error Codes from Carrier Management.
3	urn:semi-org:E125	Standard Error Codes from Equipment Self Description.
4	urn:<Supplier>-com[:<SupplierFields>]	Identification of the source of error codes.

7.3.2 *Code* — Error code is an integer value that directly maps to a particular specification error code list. The following is an example of a list of code ranges. This specification does not require the allocation of any code range by other specification. The following is just an example of codes.

Table 2 Example of Error Code Range

#	*Code Range*	*Comment*
1	1–63, 32768–65535	Standard Error Codes for Equipment - SEMI E5.
2	10000 – 10500	Equipment Supplier Specific enumerated error codes (Example).
3	5000 – 5999	Common Standardized Core Components Errors.

7.3.3 *Description* — This is a human readable description of the error code being reported. It maps directly to the definition of the error code for each of the expected result of the services as defined by the specific standard. Additional information may be supplied after this string to provide more detail about the error. Use the ":" colon symbol as delimiter (i.e. Common Component, Code 5001, "Insufficient Arguments Provided:Argument "SessionID" missing).

Table 3 Example of Error Descriptions

#	*Description*	*Comment*
1	"Unknown Object in Object Specifier"	Error description for Code "1" as defined in SEMI E5.
2	"Loadport does not exist"	Error code "49" description from E87 and defined in E5.
3	"Operation Not Supported"	Error description for Error Code "5000".

7.3.4 *Extension* — This attribute allows the supplier to include additional information regarding the error code encountered. This field is used when the description does not explicitly cover the problem and more specific information is required.

7.4 *Common Standardized Errors* — This section defines common errors to be reported by any application processing a service. Most of these errors are related to the validation of the service.

Table 4 Common Standardized Error Codes

Code	*Description*	*Comment*
5000	Operation Not Supported [:Operation specific detail]	Description of the actual error encountered. Additional information may be included.
5001	Insufficient Arguments Provided [:Operation specific detail]	One or more arguments are missing in the request.
5002	Invalid Arguments Provided [:Operation specific detail]	One or more arguments in the request are invalid or incompatible.

SEMI E139-0705
SPECIFICATION FOR RECIPE AND PARAMETER MANAGEMENT (RaP)

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

Table of Contents

List of Figures

List of Tables

1 Purpose

1.1 The purpose of this specification is to specify the cooperative interaction between the Factory Information & Control System (FICS) and the equipment to manage the specifications of equipment processing (for instance, equipment recipes).

1.2 There are six primary elements that must be supported to achieve this purpose:

- *On-tool & Off-tool Recipe Management* — Support the management of multi-part recipes within the equipment as well as at the FICS level without the requirement that these management applications be integrated.
- *Recipe Integrity* — Guarantee that the recipes that define process execution on a piece of equipment are the same as those intended for that use within the FICS.
- *Process Integrity* — Ensure that all configuration values/settings of the tool that can be changed by the user and which affect the process outcome can be managed by the host within the context of a process job.
- *Adjustable Parameter Definition* — Support the definition of recipe parameters to be used by other SEMI standards (for example SEMI E40) to adjust the parameter values in a consistent manner across all equipment.
- *On-tool & Off-tool Recipe Creation & Editing* — Define a consistent interaction protocol between the FICS and the systems(s) that support recipe creation/editing functionality.
- *Information Accessibility* — Make key information about the recipes visible to the FICS.

1.2.1 See §7 for a discussion of how the purpose is achieved. A summary is contained in ¶7.13

2 Scope

2.1 This specification defines concepts, behavior, and services to support automated management of equipment recipes and related process definition elements within a semiconductor factory. It addresses the mechanisms for interaction between semiconductor equipment, recipe editors, and the Factory Information and Control System (FICS).

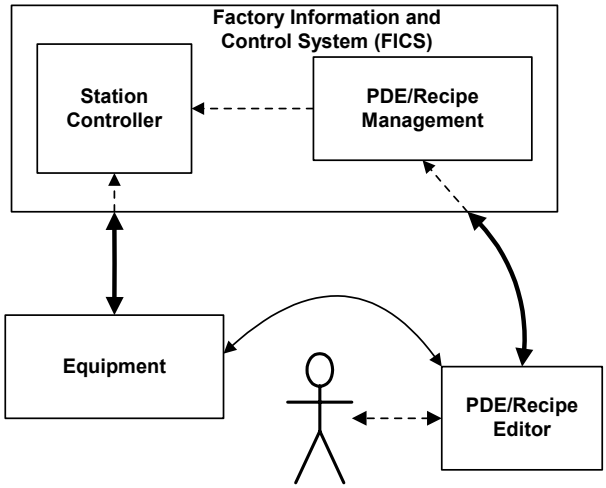

Figure 1
RaP Subject Matter

2.2 Figure 1 illustrates the interactions that are addressed in this specification. The thicker arrows are fully specified interfaces. The dashed arrows are completely unspecified interfaces that are shown here to aid in understanding the concepts of recipe management. The thinner arrow represents a partially specified interface (between equipment and *recipe editor*) that may be proprietary to the supplier. For this last interface, the messaging

details are not specified, but the required function of the interface is defined. This will be further explained later in the document.

2.3 In summary, these are the interfaces that are specified in this document:

- FICS ↔ Equipment – full messaging services
- FICS ↔ *PDE*/Recipe Editor – full messaging services
- Equipment ↔ *PDE*/Recipe Editor – functional requirements only

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 The following items are out of scope and are not specified in this document:

- The content or format of the *PDEbody*.
- The message format and communication details of message services.
- How *PDEs* are used or executed within the equipment.
- How the FICS initiates execution of *PDEs* within the equipment.
- How and when the parameter values are communicated to the equipment.
- How data is collected to document what *PDEs* and parameter values were used in a specific situation.
- The interactions between a human user and the *PDE*/Recipe Editor.
- Design of FICS applications (for example, Station Controller and *PDE*/Recipe Manager as shown in Figure 1). These applications are shown to enhance the clarity of the content of this specification.

4 Referenced Standards and Documents

4.1 *SEMI Standards*[1]

SEMI E5 — SEMI Equipment Communication Standard 2 (SECS-II)

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

SEMI E40 — Standard for Process Job Management (PJM)

SEMI E120 — Common Equipment Module (CEM)

4.2 *ISO Standards*[2]

uuid: ISO/IEC 11578:1996 Information technology - Open Systems Interconnection – Remote Procedure Call (RPC), http://www.iso.ch/cate/d2229.htm.

4.3 *OMG Standards*[3]

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

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

---

1 Semiconductor Equipment and Materials International (SEMI), 3081 Zanker Road, San Jose, CA 95134, 408.943.6900, FAX 408.428.9600

2 International Organization for Standardization, ISO Central Secretariat, 1, rue de Varembé, Case postale 56, CH-1211 Geneva 20, Switzerland. Telephone: 41.22.749.01.11; Fax: 41.22.733.34.30, Website: /www.iso.ch

3 Object Management Group, Inc., 250 First Ave. Suite 100, Needham Mass. 02494, Telephone: 781.444.0404; Fax: 781.444.0320, Website: www.omg.org

5 Terminology

5.1 *SEMI Compilation of Terms*

5.1.1 Definitions or descriptions of many of the terms used in this specification can be found in the SEMI Compilation of Terms, available on the SEMI web site, http://www.semi.org/; and in the OMG UML specification. For terms related to object models, the definition from the OMG UML specification has precedence.

5.2 *Abbreviations and Acronyms*

5.2.1 *FICS* — Factory Information & Control System

5.2.2 *PDE* — Process Definition Element

5.2.3 *RaP* — Recipe and Parameter Management – this specification.

5.2.4 *UML* — Unified Modeling Language

5.3 *Definitions*

5.3.1 *Checksum* — a single unique value calculated from a sequence of data (a file, a string, etc.) that uniquely identifies that data. It is sometimes called a "digital fingerprint" or a "message digest". No two (different) sequences of data are likely to have the same checksum. Therefore, a checksum can be used to check data integrity. A typical method is to 1) calculate the checksum for a data sequence; 2) deliver that data sequence and its checksum value to a separate entity; 3) the receiving entity recalculates the checksum value and compares it with the original from the sender. If the checksums are the same, the receiver is assured that the data has not been modified or corrupted.

5.3.2 *Factory Information and Control System (FICS)* — the software system that controls the operation of the factory and its equipment. It may include such components commonly referred to as the MES, Station Controllers, Recipe Managers, etc.

5.3.3 *Module Parameter* — settings that affect processing on an equipment module (typically a Process Module). For a Process Module, the Module Parameters might represent temperatures, pressures, flow rates, etc. An equipment recipe typically manipulates Module Parameters to accomplish its purpose.

5.3.4 *PDE (Process Definition Element)* — an executable specification of an activity or process on an equipment. The recipe for a particular equipment activity may consist of multiple *PDEs*. A *PDE* is the smallest process definition unit that can be individually managed with FICS participation or knowledge. Each *PDE* includes a *PDEheader* and a *PDEbody*.

5.3.5 *PDEbody* — the executable portion of a Process Definition Element. The *PDEbody* is typically contained within the *PDE* construct. In some cases, a *PDEbody* may exist as a separate entity, but it is always exclusively related to its *PDE*.

5.3.6 *PDEeditor* — a software system that provides the ability to create, delete, and modify a *PDE*. A *PDEeditor* may exist as a subsystem of an equipment or as a standalone system provided either by the manufacturer of the equipment or a third party.

5.3.7 *PDEheader* — the descriptive portion of a Process Definition Element. The *PDEheader* is a collection of information related to a *PDEbody*. This information is sufficient to manage and utilize the *PDE* without accessing its *PDEbody*.

5.3.8 *uuid* — universally unique identifier. This 128-bit field (often represented as a 36-character string) is calculated according to the standard ISO/IEC 11578:1996 – Remote Procedure Call (RPC). The field is guaranteed to be unique over all space and time. The uniqueness holds even though the uuids are created independently by separate entities.

5.3.9 *Variable Parameter* — a formally defined variable (setting) defined in the body of a recipe, permitting the actual value to be supplied externally.

6 Conventions

6.1 The following conventions are used in this document:

- To highlight terms specific to RaP, a term appears in *italics* within the specification. Italicized terms include class names, attributes, and services. Service names also end in (), for example "*getPDE*()".
- To prevent the definition of numerous message parameters named "XxxList," this document adopts the convention of referring to the list as "list of Xxx". In this case, the definition and data type of the parameter will be given (not the type of the list). The term "list" indicates a collection (or set) of zero or more items of the same data type.

6.2 *Object Modeling*

6.2.1 *Unified Modeling Language (UML)*

6.2.1.1 This specification uses UML notation for all class diagrams and for any object diagrams provided as examples. No other types of object modeling diagrams are used in this specification.

6.2.2 *Class Diagrams*

6.2.2.1 UML class diagrams have clearly defined meaning and are a part of this specification. Detail contained in these diagrams is not necessarily repeated in the text.

6.2.3 *Name of a Class*

6.2.3.1 The text capitalizes class names. Class attributes and services begin with lowercase letters.

6.2.4 *Abstract and Concrete Classes*

6.2.4.1 Each class is specified as Abstract or Concrete. Abstract classes are not directly implemented (that is, there are no instances). All classes defined as concrete may be directly implemented. In UML class diagrams, abstract class names are shown in *italics*.

6.2.5 *Class Attribute Definition*

6.2.5.1 The attributes of a class are defined in table format as illustrated by Table 1 below. Note that the "+" sign in front of attributes in UML class diagrams indicates "public" attributes. All attributes defined in this specification are public, therefore the "+" sign is omitted.

Table 1 Attribute Table Format

Attribute Name	*Definition*	*Access*	*Reqd*	*Form*
		RO or RW	Y or N	See list below.

6.2.5.1.1 *Access* — Attributes may be settable (ReadWrite or RW) or not settable (ReadOnly or RO) through an interface to the service provider.[4]

6.2.5.1.2 *Reqd* — Is this attribute required? Y – Yes, or N – No.

6.2.5.1.3 *Form* — Defines the data type of the attribute. Data types in this specification are high-level definitions and should be mapped to the data types of a specific technology as appropriate. In this specification, these data types are also used for the parameters of message services.

- *Binary* — A sequence of bytes that may have any value. Binary values are sometimes called "unformatted" because their structure is not apparent.
- *Boolean* — Takes the value of "true" or "false".
- *Enumeration* — A format that allows a specified list of possible values. In this document, these values are represented as text strings, but may be implemented differently (e.g. as integers that correspond to the named values).

4 The attribute table is a common format used by multiple standards. Note that all attributes defined by RaP are ReadOnly (RO).

- *Error* — The Error type is a structure that contains information about an error that has occurred. The form of the structure is implementation dependent and shall be defined by each sub-specification of RaP that describes messaging using a specific technology (for example SECS-II or XML/SOAP).
- *Integer* — A numeric value. Integers are always whole numbers. The form and number of bytes is left to the implementation definition.
- *Real* — A numeric value that may represent any whole or fractional number. The form and number of bytes is left to the implementation definition.
- *String* — A text string. Limitations on length are left to the implementation technology unless otherwise specified in this document. Strings are recommended to be implemented as UTF-8.
- *Checksum* — A checksum value calculated on a specific stream of data using a specific method of calculation.
- *Time* — Representation of the date and time of the occurrence of interest. The structure and form of items of this type is left to the implementation.
- *UUID* — Universally unique identifier created according to the ISO specification referenced in ¶4.2 and represented as ASCII characters. Note that this type is represented by all-caps "UUID" to differentiate it from the "uuid" ISO standard string definition that defines the content of an attribute of type "UUID".
- *Any* — The format may be any of the others listed in this section. Format is determined by the implementation.
- list of *xxx* — An item that can hold multiple instances of a specified type (where xxx is the data type).

6.2.6 *Association Documentation*

6.2.6.1 Associations are documented using the form of Table 2. Only Navigable Associations for the class of interest are included in the table. See ¶6.2.7 for explanation of Navigable Associations.

6.2.6.2 For each association, the table lists the Type, Description, Associated Class, Role, and Cardinality

- *Type* — The type of association. The possible values include Composition, Aggregation, and Association. See the UML standard in ¶4.3 for an explanation of these types of association.
- *Description* — Text describing the association.
- *Associated Class* — The name of the class connected by this association.
- *Role* — The role name of the associated class in the association taken from the UML diagram. Composition associations are not given roles in this document, since it usually implies containment of the aggregated objects.
- *Cardinality* — How many of the associated class may be associated with this class – documented in UML form.

Table 2 Navigable Associations for Class xxx

Type	*Description*	*Associated Class*	*Role*	*Cardinality*

6.2.7 *Association Navigability*

6.2.7.1 UML associations include the concept of "Navigability". When an association is navigable, the association may be traversed to reach the class instance at the opposite end of the association (target object).

6.2.7.2 By default, an association is navigable in both directions. If an arrowhead is shown on one end of an association, then navigability exists only in that direction.

6.2.7.3 From any object, it must be possible to obtain a reference to the target objects of any of its navigable associations. For any navigable association defined in this document, this is required of the implementer. The method of referencing the target objects is implementation dependent and is not specified in this document.

7 Overview of RaP

7.1 The RaP specification is focused on the definition, management, and transfer of equipment recipes. In RaP, the recipe components are called ProcessDefinitionElements or *PDEs*. This section describes the high level view of the concepts and the requirements that contribute to meet the purpose of this specification as described in §1.

7.2 This RaP overview makes reference to requirements defined later in the document. However, no statement in this overview section defines a requirement. This overview is intended to provide a general understanding of the RaP concepts so that the detailed requirements that follow can be more easily understood.

7.3 *RaP Participants*

7.4 RaP assumes that there are three participants in the management of recipes – the FICS, the Equipment, and the *PDEeditor*. When these provide RaP services, they become "*RaPnodes*" – the FICS becomes an *FICSnode*, the Equipment becomes an *EquipmentNode*, and the *PDEeditor* becomes an *EditorNode*.

7.5 These three participants must work together to achieve the goals of recipe management. They may exist as independent entities using defined message services (*RaPnodes*) or may be combined in some cases (as the editor is often integrated into the equipment). Since integrated modules need no standardized communications, RaP defines the services and data needed for these participants to exist independently as shown in Figure 2.

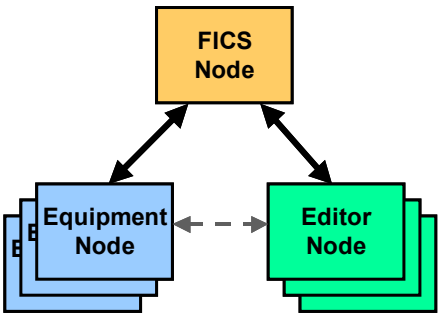

Figure 2
RaPnodes

7.6 *RaP Environment*

7.6.1 In discussing recipe management, one can divide the topic into three logically separate activities:

- Execution of recipes
- Creation and editing of recipes
- Tracking and storage of recipes

7.6.2 RaP does not specify exactly how to perform any of these three activities. However, these activities define the context in which RaP defined services and concepts are to be applied. The recipes themselves must reflect certain aspects of these activities.

7.7 *Execution of Recipes*

7.7.1 When an activity is performed on an equipment, that activity must be fully specified in order for execution to occur. RaP describes the main specification (recipe) and also parameters that can be set to amend the recipe. It does not define the process for execution of the recipe.

7.7.2 RaP assumes that the equipment or some component of the equipment represents the target for the execution of a recipe. In this document, that target is referred to as a Process Module. RaP assumes that there is a way to identify a Process Module and that certain characteristics of that Process Module are known to the factory (make, model, supplier, etc.).

7.7.3 RaP is compatible with SEMI E120. In terms of SEMI E120, the Process Module corresponds to the "AbstractModule" class. The AbstractModule class models parts of the equipment structure capable of processing material. Its attributes include "recipeType" which specifies the type of recipes that the AbstractModule can

execute. These are the recipes that RaP proposes to manage. RaP does not require use of or compliance with SEMI E120. However, if SEMI E120 compliance exists, then the source EquipmentElement of much of the RaP related data (both input and output) would be more explicitly identified.

7.7.4 From the FICS point of view, enough must be known about the recipe so that it can select the correct one for each situation (combination of machine, substrate, process step, etc.).

7.8 *Creation and Editing of Recipes*

7.8.1 RaP considers creation and editing of recipes to be an independent concept. This means that the editing capability can reside on the equipment, within the factory system, as a separate entity independent of both, or in combinations of all of these. There are advantages and disadvantages to each and these will change over time. Therefore, RaP makes no requirement on where the *PDEeditor* must reside.

7.9 *Tracking and Storage of Recipes*

7.9.1 Storage of at least some recipes on the equipment is necessary to support processing. Storage of recipes at the *PDEeditor* is also necessary, for example, to support creation of multi-part recipes. Most factories would also be expected to store a copy of most recipes for safekeeping. So all participants in recipe management might reasonably be expected to store at least some recipes.

7.9.2 RaP also makes available characteristics of the recipes that support tracking and validation (id, checksum, creation date, etc.).

7.10 *Communication Overview*

7.10.1 RaP is focused on the communications among the three "*RaPnodes*" defined above: *FICSnode*, *EquipmentNode*, and *EditorNode*. There are three types of functions that can be performed via a RaP interface:

- Transfer *PDEs* — request *PDEs* from or send *PDEs* to a node.
- Manage *PDEs* — obtain information about the *PDEs* on a node and delete them when desired.
- Verify — determine which *PDEs* are needed for a job and check that all are present and well-formed.

7.10.2 These three categories represent the eight services defined by RaP. Detailed definition of the communication services is provided in ¶8.4. That section also defines which *RaPnode*s must support which of these functions. ¶8.5 defines additional requirements on the *RaPnode*s.

7.11 *Key Recipe Concepts*

7.11.1 In order to fulfill its purpose, RaP must provide more than simple recipe transfer services. This section describes key concepts used in this specification to enhance the basic functions above to yield a robust "Recipe and Parameter Management" capability.

7.12 *ProcessDefinitionElement*

7.12.1 A recipe can be a single component or it can be made up of multiple components. The components of a recipe are called ProcessDefinitionElements or *PDEs*. The *PDE* is the central artifact of RaP. This section describes the *PDE*.

7.12.1.1 *PDE –Two Parts*

7.12.1.1.1 One barrier to automation of recipe handling is the opaque nature of the contents. For several reasons, suppliers have been reluctant to allow the user (and thus the automation system) visibility to the recipe contents. RaP addresses that problem by dividing the recipe component (or now, *PDE*) into two parts: documentation and executable. This is illustrated in Figure 3.

7.12.1.1.2 The documentation part of the *PDE* includes the *PDEheader*. This part is always public and available to the user. The *PDEheader* contains such information as the name, description, and author of the *PDE* as well as its antecedents (*PDEs* upon which this one was based) and the Process Module(s) for which it is intended. More detail about the *PDEheader* content can be found in ¶8.3.2.

7.12.1.1.3 The second part of the *PDE* contains the settings and executable instructions. The format of the *PDEbody* is not specified and is allowed to be private (that is, proprietary and opaque). Note that during job execution, the equipment relies upon the content of the PDEbody, not the PDEheader.

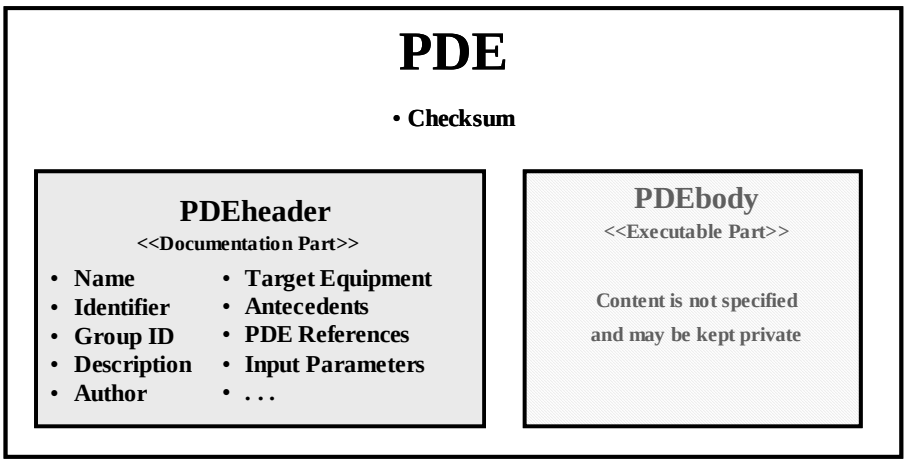

Figure 3
***PDE* Contents**

7.12.1.2 *Assuring Identity*

7.12.1.2.1 An important aspect of the *PDEheader* information is that it can provide assurance that this *PDE* is exactly the one that the factory expects it to be. This is done primarily with two data values: *uid* and *checksum*. The *uid* is a universally unique identifier that contains a uuid value as defined in the referenced ISO standard (see ¶4.2). Each time a *PDE* is modified, the *uid* of that *PDE* is required to be changed. Therefore, if the *uid* is matched, the *PDE* is guaranteed to be the expected one.

7.12.1.2.2 There are checksum values provided for the *PDE* as an additional assurance that no changes to the PDE have occurred. The checksum for a given item is calculated using the MD5 technique defined by the IETF (see §10). This computed value could be compared with the previously calculated value stored within the *PDE* to ensure that no changes have been made.

7.12.1.3 *Separate PDEbody*

7.12.1.3.1 When the *PDEbody* is contained within the *PDE*, it is of the same format[5] as the *PDEheader* and its contents are typically comprehensible to the user. When the *PDEbody* is opaque to the user or of a format that is not compatible with the header information, that *PDEbody* is stored separately from the *PDE*. This is illustrated in Figure 4. In this case, the separate *PDEbody* has its *checksum* stored in the *PDE*.

5 The definition of the format of the *PDEheader* is an implementation issue and will be defined in a subordinate specification to this one.

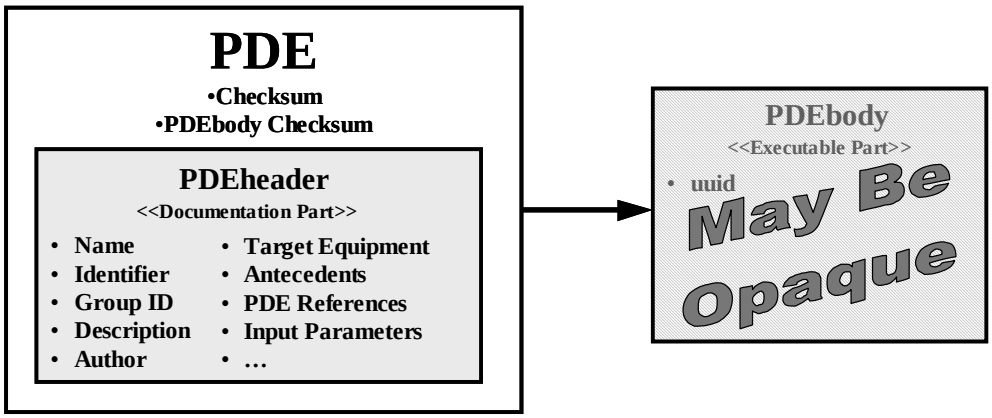

Figure 4
PDE* With Separate *PDEbody

7.12.1.3.2 Tracking of separate *PDEbodies* requires some care by the FICS, because the identity cannot be obtained from the *PDEbody* itself. Thus, a method for maintaining the relationship of the *PDE* to its body must be employed by the FICS.

7.12.1.3.3 The Equipment and *PDEeditor* can interpret the *PDEbody* contents, and so are able to resolve the relationship without such careful tracking. The *PDEbody* must include key information, such as the *uid*, that enables the equipment and the *PDEeditor* to make the association between *PDEheader* and *PDEbody* (see ¶8.5.4.5).

7.12.1.4 *Input Parameters*

7.12.1.4.1 The *PDEbody* may also define input parameters that can be set at run-time. The *PDEheader* documents these *PDEparameters*. Parameterization is discussed further in ¶7.12.3.

7.12.1.5 *Reference To Other PDEs*

7.12.1.5.1 The *PDE* may need to access other *PDEs* during execution (for example, a subroutine call). The *PDEheader* documents these references. There are two ways to reference a *PDE*. The first is by a unique identifier (*uid*). The *uid* is a reference to a specific *PDE*. The second way to reference is by a group identifier (*gid*). *PDEs* in a group can be substituted for one another and so must share the same set of input parameters. This type of reference is resolved to a specific member of the group before execution. See ¶7.12.2.5 for more on how references are resolved.

7.12.1.6 *Transition To RaP*

7.12.1.6.1 The approach for defining the *PDE* provides a good transition path from traditional proprietary recipes. Old style recipes can become the *PDEbody* when RaP is applied. Since the "*PDEbody*" can be a separate entity (in the case of proprietary content), there should be many cases where little or no modification of the original recipes will be needed. The *PDE* construct, including the *PDEheader*, acts as a wrapper.

7.12.1.6.2 There are two areas where RaP requirements may result in changes to an existing recipe structure. The first area is the requirement that an EquipmentNode be able to match a PDEheader with its external PDEbody. This would be done best by inserting the uid value into the PDEbody. However, there is a "bodyChecksum" value in the PDEheader that can be used without changing the PDEbody. This would require recalculating the checksum of the PDEbody, however. The second area is the requirement that the user be allowed to specify which PDEparameters exist. The PDEbody must contain information about the user-specified PDEparameters. However, in the case of an existing recipe with no user-specified PDEparameters, no change is necessary until the user edits the PDE to add such parameters. For example, the implementer may choose to support the old recipe format for existing recipes, but convert all new recipes to a new format that includes PDEparameter support.

7.12.2 *Recipe Structure*

7.12.2.1 When a recipe has multiple components, RaP documents the relationships among these components (see ¶7.12.1.5). This results in a hierarchical recipe structure as illustrated in Figure 5.

7.12.2.2 The recipe hierarchy must have a single *PDE* at the apex. This *PDE* is called the "*Master PDE*". It is this *PDE* that will be referenced when a job is specified. So, for example, in SEMI E40/Processing Management, the RecID (or RCPSPEC) will contain the *uid* of the *Master PDE*. See Related Information §R2-2 for more discussion of RaP support for SEMI E40 and SEMI E30. As an alternative, a *gid* may be supplied and later resolved to a *uid*.

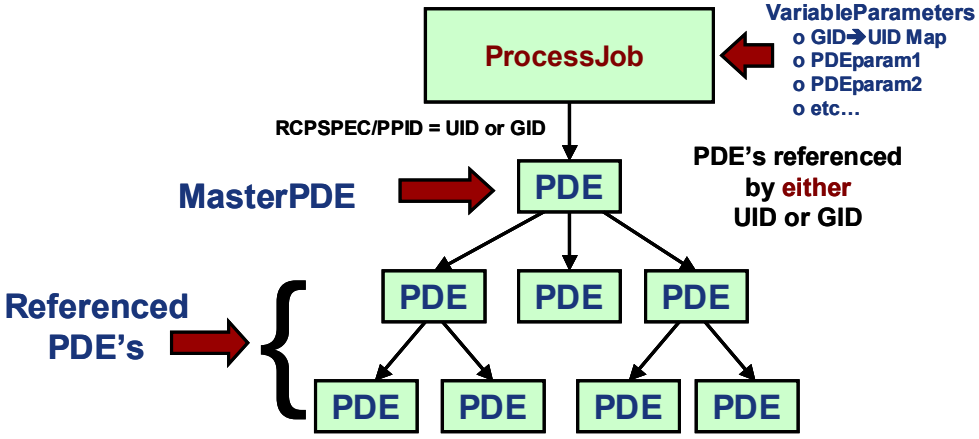

Figure 5
Recipe Structure Illustration

7.12.2.3 The recipe structure implies that the *Master PDE* is the first *PDE* to be executed. Beyond the *Master PDE*, the recipe structure does not document the order of execution of *PDEs* or the flow path of wafers through equipment. It is recommended that the equipment supplier provide a *PDE* that contains this flow information in an open format (human readable) *PDEbody*.

7.12.2.3.1 Here are a few other items of note:

- The *Master PDE* is a designation, not a type of *PDE*. A *Master PDE* must be marked as "executable" (see Table 5).
- There is no restriction on multiple references to the same *PDE*.
- The number of levels in the recipe hierarchy is not limited. However, it is to be expected that two to three levels would be sufficient in most cases. In the case where the *Master PDE* contains the entire recipe, only one level is needed.
- A *PDE* may appear in the hierarchy of multiple recipes. Such reuse of recipe components is expected and encouraged.

7.12.2.4 *Recipe Maintenance Considerations*

7.12.2.4.1 Group IDs (*gid*s) exist to make it easier to maintain multi-part recipes. It is possible to create recipes where all *PDE* references are done with *uid*s. However, this can lead to problems when a change is made to one of the components.

7.12.2.4.2 Remember that when any change is made to a *PDE*, it must become a new *PDE* with a new *uid*. In order to begin using this "new version" of the old *PDE*, some (or all) *PDEs* that referenced the original *PDE* must now reference the new one. When the referencing *PDEs* are changed to refer to the new version, each of them becomes a new *PDE*. And these have the same problem with any *PDEs* that reference them. The problem cascades to every level above the first *PDE*.

7.12.2.4.3 Any *PDE* reference that contains a *gid* can be satisfied with any member of that group. When a *PDE* is changed, it can keep the same *gid*, even as the *uid* changes. Therefore, the *PDEs* that reference the original *PDE* do not need to change. This solves the problem of cascading references.

7.12.2.4.4 Use of *gid*s as *PDE* references does add a burden, however. At some point, the *gid* must be resolved.

7.12.2.5 *Resolving PDE References*

7.12.2.5.1 ¶7.12.1.5.1 discussed the fact that *PDE* references can be either *uid*s or *gid*s. When *gid*s are used as references, they must be resolved to *uid*s before the referenced *PDE* can be executed. Resolution must occur one level at a time beginning with the *Master PDE*. Only when the specific *PDE*(s) referenced by the current level are known can one know the references these *PDEs* make to the next lower level.

7.12.2.5.2 Resolution of a *gid* to a *uid* can be performed by either the equipment or the client of the equipment. If configured to do so, the equipment will resolve any *gid*s that the client does not. If the equipment is required to resolve a *gid*, it will choose the member of that group currently at the equipment that has the newest *createDate*.

7.12.2.5.3 The client can specify *gid* resolutions for a specific processing job through a predefined Variable Parameter as input to the job. In this "*PDEmap*" parameter, the client provides a list of *gid*s with the corresponding *uid* for each. The equipment will then preferentially use the entries in this list as it resolves the recipe. If during the resolution process the equipment encounters any *gid*s that are not on the *PDEmap*, then it will attempt to resolve the *gid* as mentioned above. If the client does not supply a *PDEmap*, the equipment will resolve all *gid*s it encounters. Allowing the equipment to select the final PDEs without supervision is not advised.

7.12.2.5.4 The client can interact with the equipment to determine the best *PDEmap* values. Using the *resolvePDE*() service, the client can supply a *PDE* and receive the corresponding (fully resolved) "*outputMap*" from the equipment. The *outputMap* contains a list of the *gid – uid* pairs that result from the resolution process – the equivalent of a *PDEmap*. If desired, the client can supply an "*inputMap*" listing some *gid – uid* pairs and the equipment will use these preferentially as it resolves the *outputMap*. See the scenarios defined in Related Information 1 for examples of this interaction.

7.12.3 *Parameterization*

7.12.3.1 Parameterization of process jobs is an important concept for process control. There are other SEMI standards that define how parameters can be passed to the equipment to affect processing (for example, see SEMI E40 and SEMI E30). RaP supports the definition of those parameters. Figure 6 illustrates the flow of parameters in a RaP recipe.

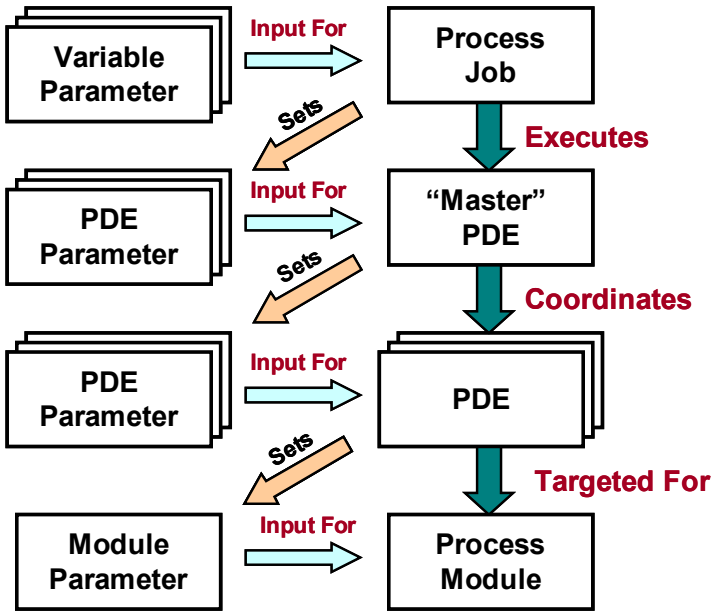

Figure 6
Parameter Flow Illustration

7.12.3.2 RaP assumes that recipes are executed as part of a larger job specification. When the "Process Job" is to be executed, settings can be passed to the job to complete its specification. In RaP, these settings are called Variable Parameters. Figure 6 shows Variable Parameters passed to the Process Job.

7.12.3.3 The Process Job references the *Master PDE* as the process recipe. During execution of the Process Job, the equipment passes Variable Parameter values to the *Master PDE* to satisfy its *PDEparameters*.

7.12.3.4 The *Master PDE* and each successive level of *PDEs* coordinate the activity of the *PDEs* at the next lower level. Each *PDE* is responsible for passing values to its referenced *PDEs* to satisfy their *PDEparameters*.

7.12.3.5 Each *PDE* can also directly set "Module Parameters", the settings that affect the Process Module (or other equipment components). Each Process Module offers a fixed set of these input parameters that can be set from within a recipe. Note that Module Parameters are not formally defined by RaP. They are used to represent internal equipment settings that recipes need to manipulate.

7.12.3.6 For a particular *PDE*, an incoming *PDEparameter* value may affect an outgoing setting of another *PDEs PDEparameter* or of a Module Parameter. This affect may be direct, where the incoming value is used as the outgoing value, or it may be indirect, where some sort of transformation is done to the incoming setting to determine the outgoing values. An example of an indirect affect is a time setting that is consumed within the *PDE* and is used to determine when a particular Module Parameter is to be set.

7.12.3.7 Notice that a *PDE* can set Module Parameters to different values during processing according to need. For example, a process chamber might have a "ChamberTemperatureSetpoint" Module Parameter. A *PDE* might require that the value for this be 450°C for the first stage of the process and 500°C for the second stage. The *PDE* could set these values directly at the proper time. However, an alternative would be to define two *PDEparameters* for this purpose: FirstStageTemp and SecondStageTemp. Each would map to ChamberTemperatureSetpoint and their default values could be set to the corresponding values. Then, when the appropriate first stage temperature is determined to be 430°C, the change can be made through a parameter setting at runtime without changing the *PDE*. In this way, multiple *PDEparameters* might map to the same Module Parameter.

7.12.4 *TransferContainer*

7.12.4.1 ¶7.10 mentions the transfer of *PDEs*. RaP provides for the definition of a process activity to span multiple, inter-related *PDEs*. To promote efficiency of messaging and organization of these *PDEs*, RaP defines the *TransferContainer*. A *TransferContainer* bundles together multiple *PDEs* for transfer.

7.12.4.2 A *TransferContainer* contains *PDEs*, *PDEbodies*, and a *Manifest*. It is convenient to conceptualize the *TransferContainer* as an archive file (for instance, a compressed "zip" file for efficient transfer) and the *PDE* and *PDEbodies* as files contained within.

7.12.4.3 There is no requirement that the *PDEs* contained in a *TransferContainer* be related in any way. Nor is there any requirement that all *PDEs* required for a particular equipment activity to be transferred together. However, RaP does insist that an external *PDEbody* always be accompanied by its *PDE (and thus, its PDEheader)*.

7.12.4.4 The *Manifest* lists which *PDEs* are included in the *TransferContainer* and associates each *PDE* with its external *PDEbody* (if one exists). The *Manifest* can also specify a storage location on the equipment for each recipe (for example a directory).

7.12.4.5 More detail is provided in ¶8.4.2.13.

7.13 *Realizing the Purpose*

7.13.1 The purpose of RaP was discussed in §1. In that section, six primary elements of the purpose were provided. This section will review these six elements and show how each is satisfied by the concepts defined in §7.

- On-tool & Off-tool Recipe Management — RaP provides the ability to uniquely identify each *PDE*. This is done independently of the versioning systems that may be used by the host or equipment recipe managers. On-tool management information is embedded in the *PDE* header (*version*, antecedent, etc.), but the host may choose to ignore this information and follow its own management procedures. Few requirements are placed on the host or equipment with regard to how management of *PDEs* is to be done.
- Recipe Integrity — To guarantee integrity of the recipes, it must be possible to uniquely identify each *PDE* and recognize that any change makes it a different *PDE*. The unique identification requirement is satisfied by the definition of a unique identifier (*uid*) containing a uuid value and by the requirement to change the *uid* with any change to the *PDE*. Checksums provide added certainty that no change was made.
- Process Integrity — RaP addresses this requirement by making all Module Parameters available for setting by a *PDE*, either directly or as input parameters to be supplied at execution time. RaP cannot guarantee that all possible parameters have been made available to the *PDEs*. This is left to the supplier and user communities to ensure.
- Adjustable Parameter Definition — RaP provides a flexible system for defining parameters for process jobs in a way that is compatible with SEMI standards (for example SEMI E40). See ¶7.12.3.
- On-tool & Off-tool Recipe Creation & Editing — RaP requires that recipe creation and editing capability exist. It defines the interfaces necessary to standardize communication with an off-tool editor. RaP does not require an off-tool editor.
- Information Accessibility — The *PDEheader* contains a large amount of user-accessible information about individual *PDEs* and how they relate to one another.

8 Requirements

8.1 *Requirements Overview*

8.1.1 This section contains all of the requirements specified by RaP. This is divided into three parts.

- The first is the Recipe Object Model, which describes the data content of the recipe components.
- The second part is RaP Services, which defines the communications between participants in recipe management.
- The third part is *RaPnode* Requirements and Clarifications, which defines the responsibilities of each recipe management participant.

8.2 *Recipe Object Model*

8.2.1 RaP recipes consist of a hierarchy of recipe components. Each recipe component is a *PDE*. In this hierarchy, each *PDE* may use other *PDEs* in order to do its job. Note that a *PDE* may appear multiple times in the hierarchy. Each appearance represents a different use of this *PDE*.

8.2.2 At the apex of the hierarchy is a single *PDE* that represents the entire recipe . This topmost *PDE* is referred to as the "*Master PDE*". The only additional RaP requirement on a PDE in order to serve as a *Master PDE* is that it shall have its "executable" attribute set to "True" (see Table 5).

8.2.3 The Recipe Object Model specifies the requirements that RaP places on the content of *PDEs*. These *PDEs* are the subject of the RaP Services (see ¶8.4). Certain information about *PDEs* can be accessed directly by *PDE* services (for example, see *getPDEdirectory()*). That information is also defined in the Recipe Object Model.

8.2.4 The Recipe Object Model specifically applies to the form of the *PDEs* during transfer. RaP does not place any requirements on the stored form of any *PDE* so long as the meaning of the *PDE* is not lost and the exact transfer form can be reconstructed for later transfer.

8.3 *PDE Class Diagram*

8.3.1 The *PDE* Class Diagram (Figure 7) represents the information contained in the *PDE*. In addition to the executable instructions, the *PDE* provides a public description of itself. This provides accessibility to the user of key information about the *PDE*, while allowing the equipment supplier to protect the integrity and any proprietary value of the actual processing instructions.

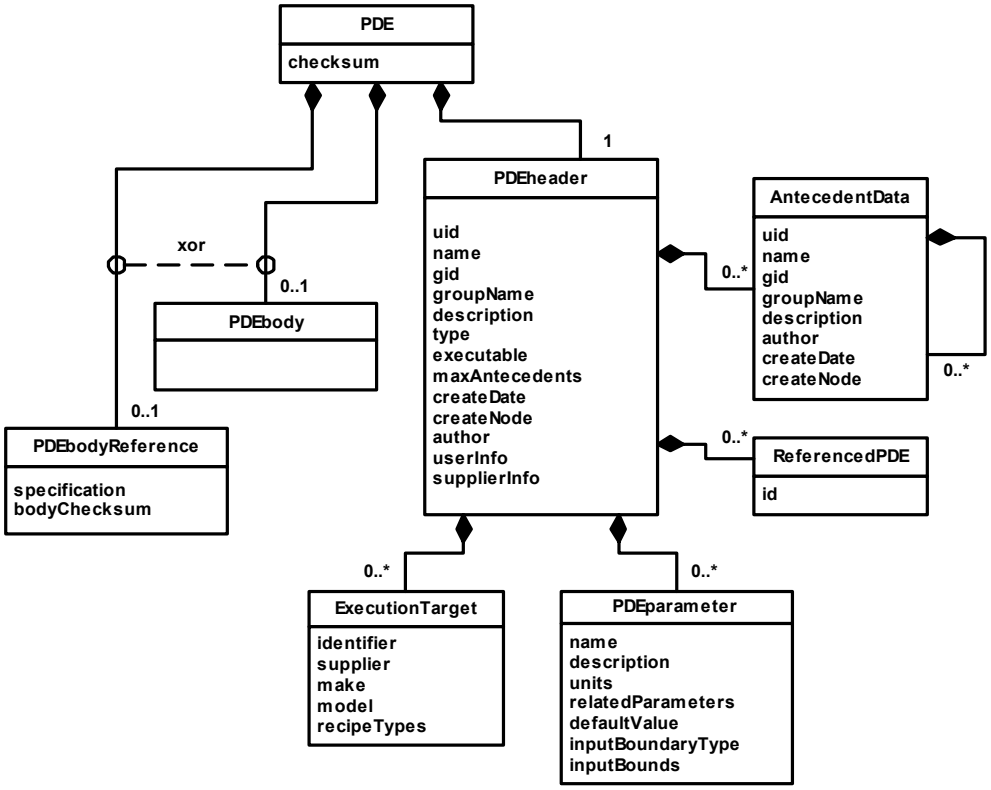

Figure 7
***PDE* Class Diagram**

8.3.1.1 *PDE Class*

8.3.1.1.1 The *PDE* includes a *PDEheader* that describes the *PDE* and a *PDEbody* that contains the executable instructions. The *PDEheader* stores key information about the *PDE*, its purpose, its history, etc.

8.3.1.1.2 The *PDEbody* may be internal to the *PDE* (as represented by the *PDEbody* class) or external (as represented by the *PDEbodyReference* class). The *PDEbody* shall be internal to the *PDE* only if its format corresponds to the same formatting rules as the *PDEheader*. The formatting rules are defined in one or more subordinate specifications of this document.

8.3.1.1.3 *PDEs* have no state. Therefore, there are no events defined for reporting the state change of a *PDE*. However, see ¶8.5.4.7 for a discussion of events relating to changes in an equipment's recipe collection.

Table 3 *PDE* Class Attributes

Attribute Name	*Definition*	*Access*	*Reqd*	*Form*
checksum	*Checksum* value of the *PDE* data structure. If the *PDEbody* is external, that external portion is not included in the checksum calculation. The checksum value is dependent on the PDE format. The subordinate specification that defines the PDE format will address practical details of checksum calculation.	RO	Y	Checksum

Table 4 Navigable Associations for *PDE*

Type	*Description*	*Associated Class*	*Role*	*Cardinality*
Composition	*PDE* contains a *PDEheader.*	*PDEheader*	none	1
Composition	*PDE* contains a *PDEbody* (XOR'ed with composition below - exactly one of these two associations will exist).	*PDEbody*	none	0..1
Composition	*PDE* contains a *PDEbodyReference.*	*PDEbodyReference*	none	0..1

8.3.2 *PDEheader Class*

8.3.2.1 The *PDEheader* documents the contents of the *PDEbody*. A *PDEheader* contains no executable instructions, but simply reflects the executable instructions contained by the *PDEbody* as well as key context and descriptive information.

8.3.2.2 A *PDEheader* shall be created by the *PDEeditor* each time a *PDE* is produced. The *PDEheader* and *PDEbody* are a matched pair and shall not be modified independently.

8.3.2.3 The *PDEheader* attributes are defined in Table 5. In addition, the *PDEheader* is composed of a list of *AntecedentData*, *ReferencedPDEs*, *PDEparameter*s, and *ExecutionTarget*s. These associations are described in Table 5 and the associated classes are described below.

Table 5 *PDEheader* Class Attributes

Attribute Name	*Definition*	*Access*	*Reqd*	*Form*
uid	Universally unique identifier, containing a uuid value as defined by ISO (see ¶4.2). The value of *uid* is not user interpretable. Any two *PDEs* that have the same *uid* shall have identical contents. They are functionally the same *PDE*.	RO	Y	UUID
name	The *name* is a user assigned designation for the *PDE*. It is not required to be unique. The value of *name* should be user comprehensible. It is recommended that the *name* reflect the purpose of the *PDE*. The *EquipmentNode* and *EditorNode* shall place no restrictions on content of the name attribute.	RO	Y	String
gid	Group identifier, formatted identically to the *uid* attribute. *PDEs* with the same *gid* may be substituted for one another in a recipe structure (see below).	RO	Y	UUID
groupName	The *groupName* is a user assigned designation for the group. All members of a group shall have the same *groupName*. However, *groupName* is not required to be unique among groups (*gid* is the unique identifier). The value of *groupName* should be user comprehensible. It is recommended that the *groupName* reflect the purpose of the *PDE* group.	RO	Y	String
description	User assigned text describing the *PDE*.	RO	Y	String

Attribute Name	*Definition*	*Access*	*Reqd*	*Form*
type	Supplier assigned string used to differentiate the kinds of *PDEs* used on the equipment. For example, the supplier might designate types such as "flow", "unit", "image", "inline-metrology", etc.	RO	N	String
executable	When = True, indicates that the *PDE* can serve as a *Master PDE*.	RO	Y	Boolean
maxAntecedents	The maximum number of *AntecedentData* records that may be kept in this *PDE*. In the case of branching (where multiple antecedents for a *PDE* existed), *maxAntecedents* is the maximum number of *AntecedentData* records in any one path backwards through the antecedents. See the definition of the *AntecedentData* class in ¶8.3.3. If zero, no Antecedents are kept.	RO	Y	Integer
createDate	Date/time the *PDE* was created.	RO	Y	Time
createNode	Identifier (*nodeID*) of the *RaPnode* where the *PDE* was created. If the creation location was not associated with a *RaPnode*, createNode shall contain an identifier that uniquely identifies the creating entity within the factory.	RO	Y	String
author	Name of the creator of the *PDE*. Note that a *PDE* can be "created", but not "modified", since any modification results in a new *PDE*.	RO	Y	String
userInfo	A list of strings that contain text entered by the user. However, neither the Equipment nor *PDEeditor* are required to interpret or react to this information.	RO	N	list of String
supplierInfo	A list of strings that contain text defined by the supplier. The *supplierInfo* strings should provide information to the user that is not included in the standard attributes.	RO	N	list of String

8.3.2.4 A *gid* is used to identify *PDEs* that are substitutable for one another. In addition to sharing the same *groupName* and *gid*, the following shall be true for all *PDEs* in the same group:

- They shall have the same set of *PDEparameters*.
- The existence of *PDEparameter* default values shall be the same (the actual default values may differ).

Table 6 Navigable Associations for *PDEheader*

Type	*Description*	*Associated Class*	*Role*	*Cardinality*
Composition	*PDEheader* may contain instances of *AntecedentData* to document the predecessors of this *PDE*.	*AntecedentData*	none	0..*
Composition	*PDEheader* may contain references to other *PDEs*.	*ReferencedPDE*	none	0..*
Composition	*PDEheader* may define *PDEparameter*s to accept externally supplied values to a *PDE* during run-time.	*PDEparameter*	none	0..*
Composition	*PDEheader* contains instances of *ExecutionTarget* that describe the equipment or module that are permitted to execute this *PDE*. If no *ExecutionTargets* are associated with the *PDEheader*, then no restriction exists. The equipment implementation may restrict the cardinality of this association to be at least one (that is, 1..*).	*ExecutionTarget*	none	0..*

8.3.2.5 Multiple *ExecutionTarget* objects are defined when multiple sets of criteria are acceptable. For example, two similar *model*s of an equipment might be able to run the same recipe.

8.3.3 *AntecedentData Class*

8.3.3.1 The *AntecedentData* class documents the predecessors of the current *PDE*. When a *PDE* is created, if a *PDE* is used as the basis for the new one, the older *PDE*'s relevant data is copied into the first *AntecedentData* instance. Then the older *PDEs AntecedentData* is appended to that of the new PDE. This provides a chronologically ordered sequence of *AntecedentData* instances. Since the antecedent *PDEs* may not be available at a later time, some of their attributes are included in the AntecedentData class.

8.3.3.2 Note that portions of multiple *PDEs* may be used to create the new one. *AntecedentData* sequences may branch as they move back in time.

8.3.3.3 The user may limit the number of *AntecedentData* instances by setting the *maxAntecedents* attribute of the *PDE*. This leaves only the most recent entries in each branch of *AntecedentData*. See Table 7 for more on *maxAntecedents*.

Table 7 *AntecedentData* Class Attributes

Attribute Name	*Definition*	*Access*	*Reqd*	*Form*
uid	Universally unique identifier of a *PDE* upon which the current *PDE* is based.	RO	Y	UUID
name	Value of the *name* attribute of the *PDE* identified by the *uid* above.	RO	Y	String
gid	Value of the *gid* attribute of the *PDE* identified by the *uid* above.	RO	Y	UUID
groupName	Value of the *groupName* attribute of the *PDE* identified by the *uid* above.	RO	Y	String
description	Value of the *description* attribute of the *PDE* identified by the *uid* above.	RO	Y	String
author	Value of the *author* attribute of the *PDE* identified by the *uid* above.	RO	Y	String
createDate	Value of the *createDate* attribute of the *PDE* identified by the *uid* above.	RO	Y	Time
createNode	Value of the *createNode* attribute of the *PDE* identified by the *uid* above.	RO	Y	String

Table 8 Navigable Associations for *AntecedentData*

Type	*Description*	*Associated Class*	*Role*	*Cardinality*
Composition	*AntecedentData* may contain *AntecedentData*. This represents the sequence of precursor *PDEs* backwards through time.	*AntecedentData*	none	0..*

8.3.4 *ReferencedPDE Class*

8.3.4.1 This class contains a reference to a *PDE* that is needed by the current *PDE* to complete its task. At any point in the structure of a recipe, the set of *ReferencedPDEs* provides the identities of the *PDEs* at the next lower level.

8.3.4.2 The value of the *id* attribute can be either a *uid* or *gid*. Since all UUID type values are universally unique, there is no conflict. If the *id* contains a *uid* value, the reference is explicit and no further work is needed. If the *id* contains a *gid* value, then it will be resolved to a *uid* at a later time (see ¶8.5.1.3 for more detail).

8.3.4.3 A *PDE* shall ensure that the *PDEparameters* of all of its *ReferencedPDEs* are satisfied, by either providing a value or allowing the default to be used (if one exists).

Table 9 *ReferencedPDE* Class Attributes

Attribute Name	*Definition*	*Access*	*Reqd*	*Form*
id	Identifier of the *PDE* being referenced. This attribute may contain the value of the *uid* or of the *gid* of an appropriate *PDE*.	RO	Y	UUID

8.3.5 *ExecutionTarget Class*

8.3.5.1 *ExecutionTarget* describes the Process Module(s) on which this *PDE* can be executed. The *supplier, make, model*, and *recipeTypes* attributes define the type of Process Module that can execute this *PDE*. Based on the definition of these attributes, an *ExecutionTarget* may represent a general set of acceptable Process Modules. If the *identifier* attribute is also be specified, the *ExecutionTarget* is limited to one specific unit. The *PDEeditor* shall ensure that the *ExecutionTarget* definition shall describe only equipment that are able to execute this *PDE*.

8.3.5.2 A *PDE's ExecutionTarget* instances should correspond to its scope of execution. A *Master PDE*'s scope would typically be the equipment.

8.3.5.3 An equipment can contain multiple Process Modules that match *ExecutionTarget* definitions for a *PDE*. In this case, the equipment, with possible guidance from the recipe, shall select which to use during processing.

Table 10 *ExecutionTarget* Class Attributes

Attribute Name	*Definition*	*Access*	*Reqd*	*Form*
identifier	Unique identifier of a specific Process Module on which this *PDE* can be executed. If the *identifier* attribute is omitted, then any equipment that matches the other attributes can execute this *PDE*.	RO	N	String
supplier	Name of the equipment supplier.	RO	Y	String
make	Classification of equipment provided by the equipment supplier. It could represent a product line, a division of the company or any other useful classification of equipment.	RO	Y	String
model	Finer grained classification of equipment provided by the equipment supplier. It could represent a tool configuration targeted at some particular wafer process or any other useful classification of equipment.	RO	Y	String
recipeTypes	This is a classification field that allows identification of recipe compatibility across multiple pieces of equipment. The supplier should define the *recipeType* values that correspond to each Process Module. Target equipment that support the same *recipeType* shall be capable of executing a *PDE* of that type without fault. While the *recipeTypes* attribute is not required by RaP, an Equipment may declare it to be required and enforce the requirement in the *PDEeditor*.	RO	N	list of String

8.3.6 *PDEparameter Class*

8.3.6.1 *PDEparameters* are inputs to a *PDE* that help determine how it will perform its task. *PDEparameter* input values are supplied by the next higher level of the recipe hierarchy. For the *Master PDE*, they are supplied by the equipment itself (as it executes the job/recipe). For all other *PDEs*, the *PDEparameter* values shall be supplied by the referencing *PDE*.

8.3.6.2 The group of *PDEs* that make up a recipe are responsible for satisfying all the settings (or Module Parameters) and any other conditions required to perform the defined activity. These Module Parameters can be set by *PDEs* at any level in the hierarchy. The value of a *PDEparameter* may affect the eventual value of one or more Module Parameters. The *PDE* may set the Module Parameters directly or it may set a *PDEparameter* that will affect the value of a Module Parameter. In either case, the affected Module Parameters are considered *relatedParameters*.

8.3.6.3 A *PDE* may set a Module Parameter to different values at different times during execution. In this case, there may be multiple input *PDEparameters* that affect the same Module Parameter.

Table 11 *PDEparameter* Class Attributes

Attribute Name	*Definition*	*Access*	*Reqd*	*Form*
name	Identifies the *PDEparameter*. It must be unique within the list of *PDEparameter*s of this *PDE*.	RO	Y	String
description	Text description of the *PDEparameter*, its intended usage, and the expected effect its settings will have on the activity.	RO	Y	String
units	The *units* of the *PDEparameter* value to be supplied.	RO	Y	String
relatedParameters	The names of the Module Parameters affected by the setting of this *PDEparameter* (if any).	RO	N	list of String
defaultValue	Value to be used for this *PDEparameter* during execution if no external input is provided.	RO	N	Any
inputBoundaryType	Specifies the type of *inputBounds* defined for this *PDEparameter*. Possible types are “List” and “Range”. If *inputBounds* are present, this attribute shall also be included.	RO	N	Enumeration

Attribute Name	Definition	Access	Reqd	Form
inputBounds	A sequence of values that represent limitations to acceptable settings for this Variable Parameter. If *inputBoundaryType* is "List", this attribute contains a set of discrete values that are acceptable for input. If the *inputBoundaryType* is "Range", then two values are included that define the acceptable range of values. In this case, the first value is the lower bound and the second is the upper bound. The acceptable range is inclusive of the *inputBounds* values.	RO	N	list of Any – matches format of *defaultValue*

8.3.6.4 The *name* of the *PDEparameter* shall be unique within the scope of the *PDE*. Also, the *name* of the Module Parameters shall be unique within the scope of a Process Module.

8.3.6.5 In the case where a *PDEparameter* maps directly to another (lower level) *PDEparameter*, the *inputBounds* defined shall be the same or more restrictive than those for the corresponding lower level *PDEparameter*. If no *inputBounds* are defined, then those for the corresponding lower level *PDEparameter* shall be used (if any are defined).

8.3.6.6 This specification does not define how or when the *inputBounds* are enforced. This is an issue of recipe execution and is out of scope for this specification.

8.3.7 *PDEbody Class*

8.3.7.1 The *PDEbody* contains settings and/or executable instructions required by the equipment to perform a run. It may be internal to the *PDE* (represented as *PDEbody* class) or external to the *PDE* (represented as *PDEbodyReference*). This section defines the former.

8.3.7.2 The *PDEbody* class is used when the content of the *PDE* instructions is visible to the user. No attributes or associations are defined for the *PDEbody*. The implementer is expected to create a subclass of *PDEbody* that represents the content of their specific *PDEbody*. The format of the *PDEbody* must follow the same formatting rules as for the *PDEheader*. These rules are defined in one or more subordinate specifications attached to this document.

Table 12 *PDEbody* Class Attributes

Attribute Name	Definition	Access	Reqd	Form
none				

8.3.8 *PDEbodyReference Class*

8.3.8.1 When the *PDEbody* format does not correspond to the same formatting rules (as defined on one or more subordinate specifications attached to this one) as the *PDEheader*, it is stored separate from rest of the *PDE*. In this case, the *PDE* has a *PDEbodyReference* to identify that separate *PDEbody*.

Table 13 *PDEbodyReference* Class Attributes

Attribute Name	Definition	Access	Reqd	Form
specification	Identifies the *PDEbody*. This is implementation dependent. For example, it might be a file specification.	RO	Y	String
bodyChecksum	The checksum value calculated for the external *PDEbody*.	RO	Y	Checksum

8.4 *RaP Services*

8.4.1 The purpose of this section is to define the message services required to support RAP functionality. Providers of RaP services are called *RaPnodes*. This section will define the different kinds of *RaPnode* and then define the services provided by each.

8.4.2 All RaP services are based on request/response semantics. Any service may be requested at any time. There are no state models associated with *RaPnodes*.

8.4.2.1 *RaPnodes*

8.4.2.1.1 RaP defines *RaPnodes* as the providers of RaP services. There are three types of *RaPnodes*: *EquipmentNodes*, *FICSnodes*, and *EditorNodes*.

- *EquipmentNode* — The *EquipmentNode* provides RaP Services for accessing the equipment.
- *FICSnode* — The *FICSnode* provides RaP Services for accessing the Factory Information and Control System (FICS).
- *EditorNode* — The *EditorNode* provides RaP Services for accessing the *PDEeditor*.

8.4.2.1.2 Figure 8 is a UML class diagram illustrating the relationships of the *RaPnodes*. As the diagram shows, there are some services that are common to all *RaPnodes*. Other services are unique to certain types of *RaPnodes*. This is more fully specified in Table 14 below.

8.4.2.1.3 The clients of RaP services can be any entity with communication access to a *RaPnode*. If the client for RaP services is also a *RaPnode*, then more robust interactions are possible with messages initiated from both *RaPnodes*. For more discussion on communication scenarios, see Related Information.

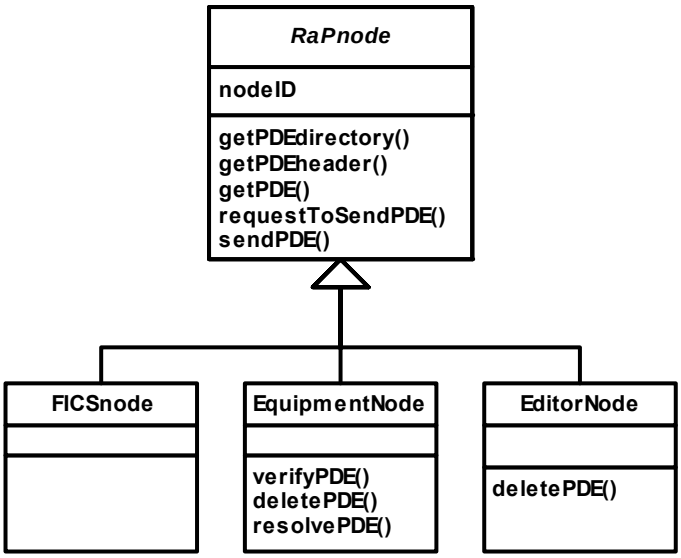

Figure 8
RaPnode Class Diagram

8.4.2.1.4 The class diagram also identifies the nodeID attribute (see Table 14). All *RaPnodes* shall define a nodeID that can be used for identification in communications. The nodeID is referenced by the *createNode* attribute of the *PDEheader* class.

Table 14 *RaPnode* Class Attributes

Attribute Name	*Definition*	*Access*	*Reqd*	*Form*
nodeID	Identifier of the *RaPnode*. The nodeID shall be unique within the factory where it is being used.	RO	Y	String

8.4.2.2 *Required Services Per Node*

8.4.2.2.1 As discussed in §2, RaP defines services that apply to three different communication nodes: the FICS, the *Equipment*, and the *PDEeditor*. Each of those nodes is required to provide certain services to be compliant to RaP. Table 15, Service Descriptions, describes each service defined by RaP and which node must provide that service. This listing corresponds to the services listed in Figure 8.

8.4.2.2.2 The Service Descriptions table has six columns:

- The first column lists the services.
- The second column defines the type of service. This may be a request/response message pair (denoted by "R") or a notification message (denoted by "N").
- The third column is a textual description of the service.
- Columns 4–6 each represent a RaP communication node. For each service, a checked box in this column indicates that the node is required to provide this service. An empty box indicates that the service shall not be provided.

Table 15 Service Descriptions

Service Name	*Type*	*Description*	*Equipment*	*FICS*	*Editor*
getPDEdirectory()	R	This service requests a list of the *PDEs* managed by the *RaPnode*. The requestor may supply selection criteria for which *PDEs* to report and may also select certain attributes to be returned for each reported *PDE*.	☑	☑	☑
deletePDE()	R	This service requests the service provider to delete the specified *PDE*(s) from the *RaPnode*.	☑	☐	☑
getPDEheader()	R	This service requests the service provider to return the complete header of the specified *PDE*(s).	☑	☑	☑
getPDE()	R	This service requests the service provider to return the specified *PDE*(s) in a *TransferContainer*.	☑	☑	☑
requestToSendPDE()	R	This service requests the service provider to grant or deny permission to transfer a collection of *PDEs* in a *TransferContainer* to the service provider.	☑	☑	☑
sendPDE()	R	This service requests the service provider to accept the set of *PDE*'s in the included *TransferContainer*.	☑	☑	☑
resolvePDE()	R	This service requests the service provider to determine the complete recipe hierarchy beneath the specified *PDE* based on the *ReferencedPDEs*.	☑	☐	☐
verifyPDE()	R	This service requests the service provider to check various aspects of the specified *PDE*.	☑	☐	☐

8.4.2.3 *Service Message Parameters*

8.4.2.3.1 The following service parameter dictionary (Table 16) defines the name, description, and format for parameters used by RaP services. The format column uses data types defined in ¶6.2.5.1.3.

Table 16 Service Parameter Dictionary

Parameter Name	*Form*	*Description*	*Where Used*
delRspInfo	list of (*uid*, *delRspStat*)	Contains a list of the status for each specified *PDE* (as identified by the *uid*).	*deletePDE*()

Parameter Name	*Form*	*Description*	*Where Used*
delRspStat	Error	Status response for the *deletePDE*() service. The following responses are allowed: "OK" — Successfully deleted specified *PDE*. "PDEnotFound" — *PDE* was not found on the service provider. "PDElocked" – The PDE is present, but cannot be deleted at this time. For example, the *PDE* might be executing or reserved for a job. The *PDE* was not deleted. "Other" — A problem occurred that is not described by the other enumerated values. The *PDE* was not deleted.	*delRspInfo* parameter
dirRspStat	Error	Status response for the *getPDEdirectory*() service. The following responses are allowed: "OK" — Successfully returned requested data. "BadFilter" — One or more of the *PDE* filter specified were not properly specified. "BadAttribute" — One or more of the *PDE* attributes requested to be returned do not exist.	*getPDEdirectory*()
getRspInfo	list of (*uid*, *getRspStat*)	Contains a list of the status for each specified *PDE* (as identified by the *uid*).	*getPDE*(), *getPDEheader*()
getRspStat	Error	Status response for the *getPDE*() service. The following responses are allowed: "OK" — Successfully returned the *PDE* (or header). "PDEnotFound" — The *PDE* was not available from the service provider. "PDElocked" — The *PDE* exists, but cannot be transferred now (for example, it may be in use). "Other" — A problem occurred that is not described by the other enumerated values.	*getRspInfo* parameter
inputMap	list of (*pdeRef*, *resolution*)	Client provided list of *PDE* references and corresponding resolutions. Each *inputMap* entry is a pair of values: a reference to a *PDE* and the *uid* that is the resolution for that reference. When the *pdeRef* is a *uid*, the *resolution* shall always contain the same value. This list may contain zero entries.	*resolvePDE*(), *verifyPDE*()
operator	Enumeration	Choice of available *operators* that relate the attribute value to the specified value. The following values are allowed: "EQ" — Equals (numeric or string) "GT" — Greater Than (numeric) "LT" — Less Than (numeric) "GE" — Greater than or equal to (numeric) "LE" — Less than or equal to (numeric) "Like" — Contains the substring (string) "NotEQ" — Not Equal (numeric or string) "NotLike" — Does not contain the substring (string)	*PDEFilter* parameter
outputMap	list of (*pdeRef*, resolution)	Service provider generated list of *PDE* references and their resolutions. Each *outputMap* entry is a pair of values: a reference to a *PDE* and the *uid* that is the resolution for that reference. When the *pdeRef* is a *uid*, the *resolution* shall always contain the same value.	*resolvePDE*()