9.1.2.3.1 *GetStateMachines Operation Arguments*

Table 8 GetStateMachines Argument Definitions

Argument	*Description*	*Kind*	*Form*
stateMachines	All state machine metadata provided by the equipment.	out	Unordered list of elements of type StateMachine, described in Section 10.8 .

9.1.2.4 *GetSEMIObjTypes* — Upon receiving this request, the equipment shall return all OSS ObjType metadata provided by the equipment.

9.1.2.4.1 *GetSEMIObjTypes Operation Arguments*

Table 9 GetSEMIObjTypes Argument Definitions

Argument	*Description*	*Kind*	*Form*
objTypes	All SEMI ObjType metadata provided by the equipment.	out	Unordered list of elements of type SEMIObjType, described in Section 10.8 . If no SEMI ObjTypes are provided, this list shall be empty.

9.1.2.5 *GetExceptions* — Upon receiving this request, the equipment shall return all exception metadata provided the equipment.

9.1.2.5.1 *GetExceptions Operation Arguments*

Table 10 GetExceptions Argument Definitions

Argument	*Description*	*Kind*	*Form*
exceptions	All exception metadata provided by the equipment.	out	Unordered list of elements of type Exception, described in Section 10.7 .

9.1.2.6 *GetEquipmentStructure* — Upon receiving this request, the equipment shall return all equipment structural metadata provided by the equipment.

9.1.2.6.1 *GetEquipmentStructure Operation Arguments*

Table 11 GetEquipmentStructure Argument Definitions

Argument	*Description*	*Kind*	*Form*
equipmentStructure	All equipment structural metadata provided by the equipment.	out	Structured data, of a type derived from the SEMI E120 type Link or EquipmentElement, that represents the equipment as a whole.

9.1.2.7 *GetEquipmentNodeDescriptions* — Upon receiving this request, the equipment shall return all requested equipment node description metadata defined for the equipment. If the request includes unrecognized equipment node identifiers, the equipment shall return only node descriptions for recognized identifiers. All unrecognized identifiers shall be returned with the response.

9.1.2.7.1 *GetEquipmentNodeDescriptions Operation Arguments*

Table 12 GetEquipmentNodeDescriptions Argument Definitions

Argument	*Description*	*Kind*	*Form*
equipmentNodeIds	The id's of the requested equipment node descriptions	in	List of text, each entry equal to a valid SEMI E120 Locator value (see E120, Related Information 2). If an empty list is provided, the equipment shall return node descriptions for all equipment nodes.
equipmentNodeDescriptions	All equipment node description metadata.	out	Structured data of type NodeDescriptionResult.

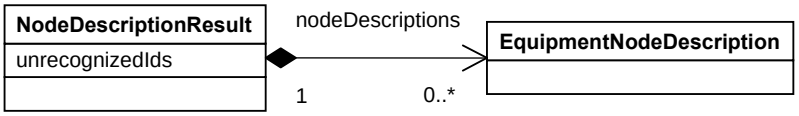

Figure 4
Interface for Accessing Equipment Metadata

9.1.2.8 *NodeDescriptionResult* — This class provides the list of requested EquipmentNodeDescription elements and any unrecognized equipment node id's that may have been provided with the request.

9.1.2.8.1 *NodeDescriptionResult Attribute Definition Table*

Table 13 NodeDescriptionResult Attribute Definition

Attribute Name	*Definition*	*Form*
unrecognizedIds	Any unrecognized equipment node identifiers provided in the original request.	List of text, each entry equal to an unrecognized equipment node identifier provided in the GetEquipmentNodeDescriptions request.

9.1.2.8.2 *NodeDescriptionResult Association Definition Table*

Table 14 NodeDescriptionResult Association Definition

Association Role Name	*Definition*	*Comments*
nodeDescriptions	The requested equipment node description metadata.	Zero or more elements of type EquipmentNodeDescription. The equipment shall return node description metadata for all recognized equipment node identifiers. If the same node identifier appears more than once in the request, the equipment shall return only one corresponding node description.

9.1.2.9 *GetLatestRevision* — Upon receiving this request, the equipment shall return the most recent date and time at which the metadata contents were changed. The equipment shall store the date and time of the most recent change in non-volatile memory. The equipment shall not generate false revision dates, in which there is no change in the available metadata, but the latest revision date/time is changed.

9.1.2.9.1 *GetLatestRevision Operation Arguments*

Table 15 GetLatestRevision Argument Definitions

Argument	*Description*	*Kind*	*Form*
revision	The most recent date and time at which the metadata was changed.	out	Structured data of type MetadataRevision, described in Section 9.1.2.11 .

9.1.2.10 *NotifyOnRevisions* — Upon receiving this request, the equipment shall either enable or disable metadata revision notices to the requesting client, based on the operation arguments provided by the client. The equipment shall store the client's preference in non-volatile memory.

9.1.2.10.1 *NotifyOnRevisions Operation Arguments*

Table 16 NotifyOnRevisions Argument Definitions

Argument	*Description*	*Kind*	*Form*
notification	Specifies the client's notification preference.	in	Structured data of type NotificationRequest, described in Section 9.1.2.12 .

9.1.2.10.2 Clients must invoke this operation with the 'enable' attribute of the notification argument set to 'true' in order to receive notifications. The default behavior for revision notifications for a given client is disabled. The equipment shall automatically disable notification for a client if the equipment is in an enabled communications state and determines that the client is no longer accessible to receive such notifications. In this case, the client must invoke this operation in order to re-enable notifications.

9.1.2.10.3 If the 'enable' attribute of the notification argument provided is 'true', the equipment shall begin notifying the client (if it is not already doing so) whenever a change in the metadata is detected, starting with the first detected change after the client invokes the operation. If the change is detected while the equipment is in a disabled communications state, the equipment shall notify the interested clients at the first opportunity when its communication status permits.

9.1.2.10.4 If the 'enable' attribute of the notification argument provided is 'false', the equipment shall cease sending revision notifications to the client, if it is currently doing so. The equipment shall ignore duplicate enable/disable requests from the same client (see limitations, Section 3).

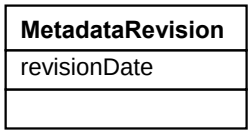

Figure 5
Metadata Revision

9.1.2.11 *MetadataRevision* — This class describes the most recent metadata revision date and time.

9.1.2.11.1 MetadataRevision Attribute Definition Table

Table 17 MetadataRevision Attribute Definition

Attribute Name	*Definition*	*Form*
revisionDate	The most recent date and time at which the equipment metadata has changed.	Text, formatted according to ISO 8601. Implementations may specify additional formatting restrictions within the 8601 specification.

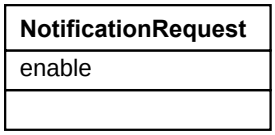

Figure 6
Notification Request

9.1.2.12 *NotificationRequest* — This class describes the notification preferences of the metadata client.

9.1.2.12.1 NotificationRequest Attribute Definition Table

Table 18 NotificationRequest Attribute Definition

Attribute Name	*Definition*	*Form*
enable	The client's preference for revision notification.	Boolean. If true, the client's preference is to be notified of metadata revisions. If false, the preference is to not be notified of metadata revisions.

9.2 *Metadata Client*

9.2.1 Figure 7 shows the interface and argument data types that clients must support in order to receive metadata revision notifications.

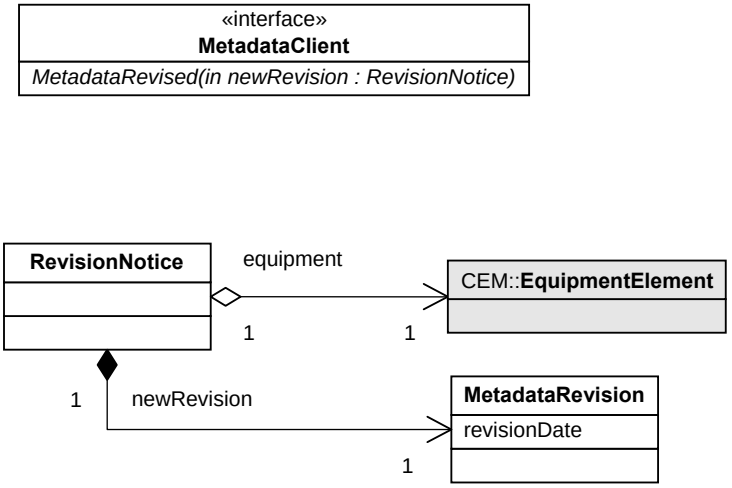

Figure 7
Metadata Client Interface

9.2.2 *MetadataClient Operations*

Table 19 MetadataClient Operation Definition

Operation	*Description*	*Type*
MetadataRevised	Notifies the client that the metadata has recently changed.	FF

9.2.2.1 *MetadataRevised* — The equipment shall send this notification whenever it detects a change in the metadata, if the client has requested to be notified through a prior invocation of the "NotifyOnRevisions" operation of the EquipmentMetadataManager interface.

9.2.2.1.1 MetadataRevised Operation Arguments

Table 20 MetadataRevised Argument Definitions

Argument	*Description*	*Kind*	*Form*
revision	The date and time at which the metadata change was detected by the equipment.	in	Structured data of type RevisionNotice, described in Section 9.2.2.2 .

9.2.2.2 *RevisionNotice* — This class provides the date and time of any metadata revision, and the equipment whose metadata was changed.

9.2.2.2.1 RevisionNotice Association Definition Table

Table 21 RevisionNotice Association Definition

Association Role Name	*Definition*	*Comments*
equipment	The root equipment node that represents the equipment as a whole.	One and only one element of a type derived from the SEMI E120 type EquipmentElement.
newRevision	The date and time at which the metadata was revised.	Structured data, of type MetadataRevision

9.2.2.2.2 *equipment* — Providing the root equipment node with a RevisionNotice should not be interpreted to mean that the root equipment node's metadata has changed. It is strictly provided as a means for clients to determine which equipment has undergone the change.

10 Equipment Metadata

10.1 *Management of Equipment Metadata*

10.1.1 When equipment configuration changes result in a change to any information described using the classes in Section 9.1.2.9 of this document or by the SEMI Common Equipment Model, the equipment shall update the metadata description accordingly at the time of the change. If clients have requested revision notification (see Sections 9.1.2.9 and 9.1.2.10), the equipment shall notify each interested client at the time of the change, or at the first opportunity when its communication status permits. This includes reconfiguration of the equipment after its introduction into the factory. Ensuring that all aspects of equipment metadata are kept up-to-date and available to the factory is a critical aspect of maintaining equipment metadata.

10.2 *Equipment Structure*

10.2.1 The physical equipment structure shall be described according to the SEMI Common Equipment Model. Suppliers shall provide a model of those elements of their equipment for which it is necessary and practical to facilitate (at least) utilization tracking, process control, and technical support objectives.

10.3 *Equipment Nodes*

10.3.1 *Equipment Node Description*

10.3.1.1 Figure 8 shows how to describe the parameters, events, objTypes, and exceptions available from an element (or node) of the equipment hierarchy. Providing this descriptions indicates that the specified node acts as the source of all Events, Exceptions, ObjTypes, and Parameters that are included in the description. All Parameters, Exceptions, ObjTypes, and Events associated with a given equipment node are independent of the Events, Exceptions, ObjTypes, and Parameters associated with other equipment nodes.

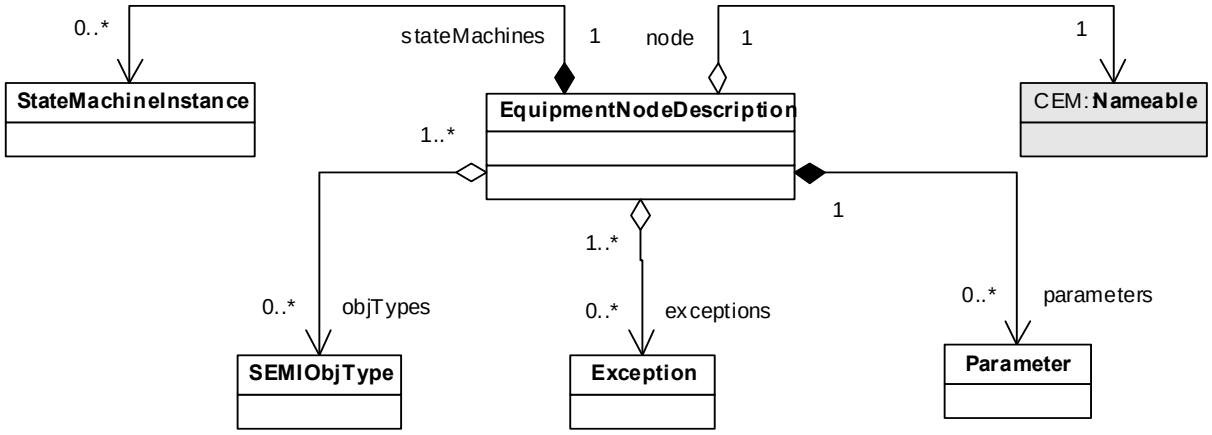

Figure 8
Describing Equipment Nodes

10.3.1.2 *EquipmentNodeDescription* — This class is used to identify a specific element of the equipment hierarchy, and to describe the Parameters, Exceptions, ObjTypes, and Events for which this node is the source. This is achieved through the StateMachineInstance (Section 10.8.6), SEMIObjType (Section 10.8.5), Exception (Section 10.7), and Parameter (Section 10.4) classes. Only one EquipmentNodeDescription shall be provided for a given equipment node. Equipment elements may be any concrete class derived from Nameable, as defined in the SEMI E120 specification, Common Equipment Model.

10.3.1.3 *EquipmentNodeDescription Association Definition Table*

Table 22 EquipmentNodeDescription Association Definition

Association Role Name	*Definition*	*Comments*
stateMachines	An unordered list of state machines that are supported by this equipment node.	List of elements of type StateMachineInstance, described in Section 10.8.6. There shall be one map provided for each event that the node can generate, including events for which no Parameters are available.
objTypes	An unordered list of the SEMIObjTypes that are associated with this equipment node.	List of elements of type SEMIObjType, describe in Section 10.8.5 .
exceptions	An unordered list of exceptions that can be reported by this equipment node.	List of elements of type Exception described in Section 10.7 .
parameters	An unordered list of typed parameters provided by this equipment node.	List of elements of type Parameter, described in Section 10.4 . Any parameters that represent variables from SEMI standards shall be typed and named according to Sections 10.5.4 and 10.4.10 . Each parameter name for this node shall be unique. Parameter names need not be unique across equipment nodes.
node	The equipment node being described.	Element of a type derived from the SEMI E120 Nameable class.

10.4 *Parameters*

10.4.1 Parameters can represent any concept that can be described by any of the primitive or composite types described in Section 10.5 . A parameter can generally be thought of as a named, typed field that has a specific meaning, utility or other information value. Examples are process control variables, equipment configuration settings, measurement results, wafer maps, data variables, etc. Parameters can only be defined by an equipment node (Section 10.3), a SEMI ObjType (Section 10.9), or an exception (Section 10.7).

10.4.2 Figure 9 shows how to describe named, typed parameters with units, constraints, and associations with other parameters. Type and units information is provided through the ParameterTypeDefinition referred to by the DefinedType class. ParameterTypeDefinition is described in Section 10.5.1 .

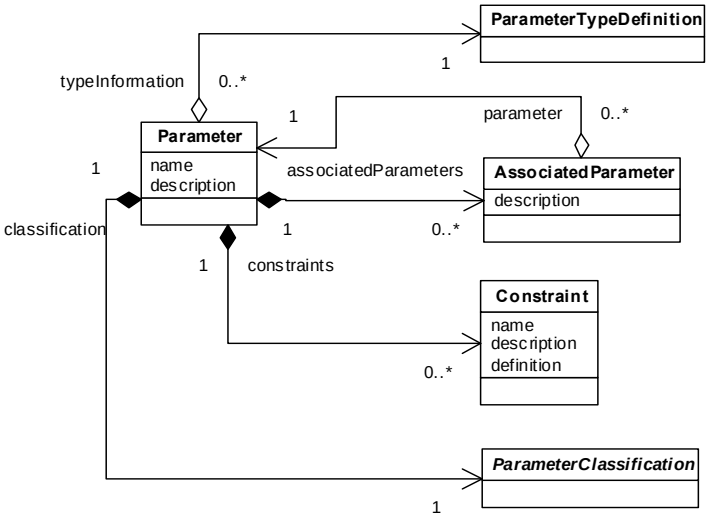

Figure 9
Describing Typed Parameters

10.4.3 Suppliers shall provide a model of those Parameters for which it is necessary and practical to facilitate (at least) utilization tracking, process control, and technical support objectives.

10.4.4 *Parameter Attribute Definition Table*

Table 23 Parameter Attribute Definition

Attribute Name	*Definition*	*Required*	*Form*
name	The name of the parameter.	Y	Text.
description	A human-readable description of the purpose or usage of the Parameter.	Y	Text.

10.4.5 *Parameter Association Definition Table*

Table 24 Parameter Association Definition

Association Role Name	*Definition*	*Comments*
constraints	A description of any restrictions associated with this parameter.	List of structured data, of type Constraint, described in Table 26 . Logical AND semantics apply between constraints if more than one is described.
associatedParameters	List of other parameters in the system that are related to this parameter.	List of structured data, of type AssociatedParameter, described in Section 10.4.7 .
typeInformation	Refers to a type definition that describes the type and units of this parameter.	One and only one element of type ParameterTypeDefinition.
classification	Describes the nature of the Parameter	One and only one element of any type derived from ParameterClassification, described in Section 10.4.11.2 .

10.4.6 Each parameter can provide a list of zero or more associated parameters, and a human-readable description of the meaning of the relationship.

10.4.7 *AssociatedParameter* — This class can be used to describe a generic association with another parameter anywhere in the equipment hierarchy. If such an association is created, this indicates that the Parameter being defined is related in some way to the parameter referenced by this class. The meaning of the relationship is documented in the description attribute of the AssociatedParameter class. More specific meaning can be attributed to an associated Parameter via the classification types described in Section 10.4.11 .

10.4.7.1 *AssociatedParameter Attribute Definition Table*

Table 25 AssociatedParameter Attribute Definition

Attribute Name	*Definition*	*Required*	*Form*
description	Description of the relationship between the defined parameter and the parameter identified by this association.	Y	Text.

10.4.7.2 *AssociatedParameter Association Definition Table*

Table 26 AssociatedParameter Association Definition

Association Role Name	*Definition*	*Comments*
associatedParameters	A list of parameters that are associated with this parameter.	Unordered list of structured data, of type Parameter.

10.4.8 *Constraint* — Represents a constraint associated with this parameter. A constraint can be used to describe limitations on the valid values of a parameter, the rate at which the parameter can be collected for off-tool reporting, or any other similar restriction on the value or usage of the parameter.

10.4.8.1 *Constraint Attribute Definition Table*

Table 27 Constraint Attribute Definition

Attribute Name	*Definition*	*Required*	*Form*
name	The name of the constraint.	Y	Text.
description	A description of the purpose or meaning of the constraint.	Y	Text.
definition	The definition of the constraint.	Y	Text, formatted according to Section 10.4.9 .

10.4.9 *Constraint Definition*

10.4.9.1 Parameter constraints describe restrictions on the valid values of Parameters. A constraint is documented as a propositional formula, which evaluates TRUE with valid parameter values. Relational algebra defines a rich syntax for propositional formulas; however, this syntax is not commonly used for commercial applications. Standard Query Language (SQL) is a well-documented commercial application of relational algebra well suited to this application. Specifically, the syntax of the **WHERE** clause (of the SQL **SELECT** statement) is suitable for describing constraints on Parameter values. The **WHERE** clause of the SQL **SELECT** statement is a propositional-expression that consists of one or more comparisons of values with constants. It has the following form.

WHERE propositional-expression [nested-propositional-expression];

For the purposes of this specification, a subset of the SQL WHERE syntax (the complete syntax is discussed in Related Information, Section R2-1) shall be used to define constraints. Specifically, the IN, BETWEEN, and IS operators are not supported in constraint definitions.

10.4.9.2 For example, a Parameter named "TemperatureSetpoint" that represents a temperature set point that may not be less than 100 or greater than 200 may be described using the following SQL **WHERE** clause.

WHERE TemperatureSetpoint > 100 **AND** TemperatureSetpoint < 200;

10.4.9.3 *Describing Reporting Frequency Constraints*

10.4.9.3.1 To describe the rates at which a given Parameter can be reported for data acquisition, a specific convention using the subset of the SQL **WHERE** clause described in Section 10.4.9.1 shall be used. For Parameters where this is a consideration, a constraint must be provided named "ReportingPeriod". The definition of the ReportingPeriod constraint shall take the following form:

WHERE *parameterName*.ReportingPeriod *comparison*;

10.4.9.3.2 Here, *parameterName* is the name of the Parameter to which the constraint applies, and *comparison* is any legal combination of terms, operators, or constants necessary to describe the constraint on valid reporting periods for the Parameter. The time division for any constants that refer to the reporting period shall be in seconds. If the constraint definition is used to express an integer multiple of a fundamental reporting period, the variable 'n' shall be used to represent the multiplier.

10.4.9.3.3 For example, the constraint definition below specifies that the supported reporting period for "Temperature" is any continuous value between 0.01 seconds and 60 seconds. Additional examples are provided in Related Information, Section R4-1.

WHERE Temperature.ReportingPeriod >.01 AND Temperature.ReportingPeriod < 60;

10.4.10 *Describing SEMI Standard Variables and/or Attributes*

10.4.10.1 If the supplier is using a Parameter to represent variables or attributes that have been defined in a SEMI standard, the name attribute of the Parameter class shall be equal to the name of the variable or attribute exactly as it is spelled, using the same case convention as found in the corresponding standard. The type mapping described in Section 10.5.4 shall determine the type assigned to the Parameter.

10.4.11 *Classifying Parameters*

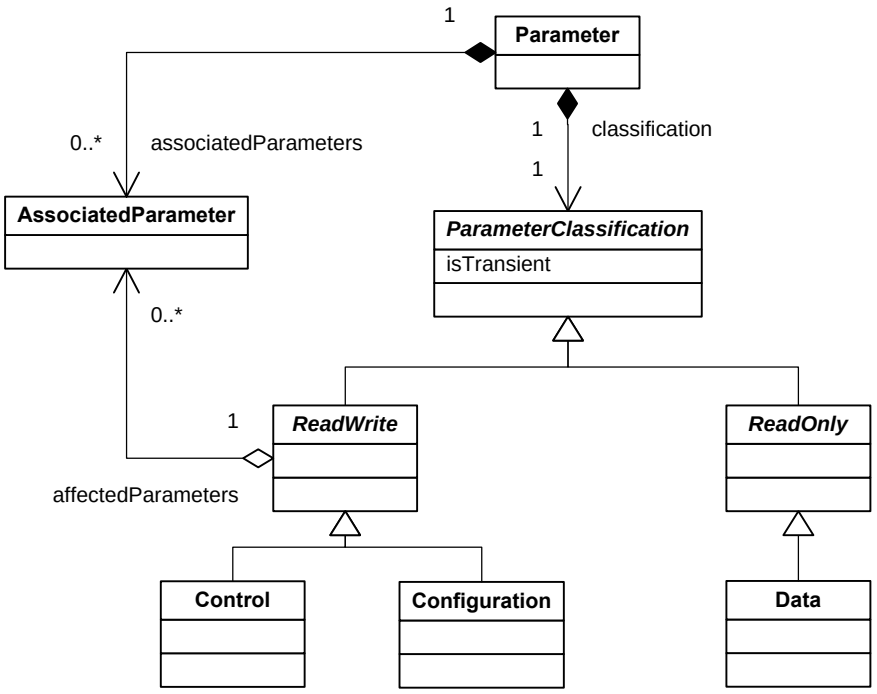

Figure 10
Parameter Classifications

10.4.11.1 Figure 10 shows the classification scheme used to describe the characteristics of a Parameter. There are two primary classifications, read-only and read-write, that specify whether or not the parameter's value can be set by an external entity. If read-only, the Parameter is classified as data. If read-write, there are several subclasses that can be used to further refine the description of the Parameter's function.

10.4.11.2 *ParameterClassification* — This class is the abstract base class for all Parameter classifications.

10.4.11.3 *ParameterClassification Attribute Definition Table*

Table 28 ParameterClassification Attribute Definition

Attribute Name	*Definition*	*Form*
isTransient	Whether or not the Parameter's value is only accessible or meaningful under certain conditions.	Boolean. See Section 10.4.11.3.1.

10.4.11.3.1 *isTransient*

10.4.11.3.1.1 If false, the Parameter's value is guaranteed to be accessible and meaningful at all times that the equipment is available and in a state in which it is possible to communicate with external entities. Parameters that correspond to SEMI E30 equipment constants and status variables, for example, are non-transient.

10.4.11.3.1.2 If true, the Parameter is said to be "transient". The value of a transient Parameter is not guaranteed to be accessible or meaningful except under certain conditions. For example, Parameters that correspond to SEMI E30 "data values" are transient. Additionally, a Parameter may represent an attribute of a transient SEMI E39 ObjType, in which case its accessibility is determined by the lifecycle of the ObjType. The Parameter may provide data that is only available when the equipment (or SEMI E39 ObjType) is in a specific state or states. If a transient Parameter is available for reporting with a specific event, its value may only be meaningful while the equipment is in the target state for which the event was reported. Or, the Parameter's value may only be meaningful at the time the event

occurs. Providing a mechanism to describe the lifecycle for transient Parameters using metadata is beyond the scope of this specification.

10.4.11.4 *ReadWrite* — This abstract class identifies a Parameter as being settable by an external entity. The settings for such Parameters may influence the values or dynamics of other Parameters. For example, Parameters corresponding to E30 "equipment constants" may be set by an external entity, and so would be classified as ReadWrite. Parameters corresponding to E30 "status variables" or "data values" cannot be set by external entities, so would not be classified as ReadWrite.

10.4.11.5 *ReadWrite Association Definition Table*

Table 29 ReadWrite Association Definition

Association Role Name	*Definition*	*Comments*
affectedParameters	A list of associated Parameters that are affected by the settings of this Parameter.	Unordered list of structured data, of type AssociatedParameter.

10.4.11.5.1 *affectedParameters*

10.4.11.5.1.1 If the Parameter's settings directly affect the values or dynamics of other Parameters elsewhere in the metadata, an AssociatedParameter instance must be provided and described for each such relationship for the Parameter being defined. Other generic associations may exist for the Parameter, but the specific associations that arise from the settings of the Parameter's value must be accessible through the affectedParameters association provided by this base class unless otherwise specified in this document.

10.4.11.6 *ReadOnly* — This abstract class identifies a Parameter as being un-settable by an external entity. Its value, however, is accessible to external entities. This class has no attributes or associations. For example, Parameters corresponding to E30 "status variables" or "data values" cannot be set by an external entity, and so would be classified as ReadOnly. Parameters corresponding to E30 "equipment constants" can be set by external entities, so could not be classified as ReadOnly.

10.4.11.7 *Data* — This class is derived from ReadOnly, and identifies a Parameter as providing data values. If non-transient, (for example, Parameters corresponding to E30 "status values"), the data values are accessible and meaningful at all times. If transient, (for example, Parameters corresponding to E30 "data values"), the data values are not guaranteed to be meaningful except when they are explicitly provided with the occurrence of specific events or, if they represent attributes of an SEMI E39 ObjType, when the corresponding ObjType has at least one active instance. This class defines no attributes or associations.

10.4.11.8 *Control* — This is derived from ReadWrite, and identifies a Parameter as being settable by an external entity (for example, Parameters corresponding to recipe settings). Changes in the values of a Control Parameter affect the way in which the equipment processes, measures, or tests material and may affect the values of other Parameters. This class defines no attributes or associations.

10.4.11.8.1 If other Parameters are affected by this Parameter's settings, all such affected Parameters shall have a corresponding AssociatedParameter instance provided with the Control Parameter's definition. Each such AssociatedParameter instance shall be accessible via the "affectedParameters" association of this class (inherited from ReadWrite).

10.4.11.8.2 If non-transient, the values of this Parameter are accessible and meaningful at all times, and can be set by an external entity under conditions determined by the equipment.

10.4.11.8.3 If transient, the values of this Parameter are not guaranteed to be meaningful or settable at all times. If the Parameter represents an attribute of an SEMI E39 ObjType, the Parameter will only be settable when the corresponding ObjType has at least one active instance.

10.4.11.9 *Configuration* — This class is derived from ReadWrite, and identifies a Parameter as being settable by an external entity, and in some way influencing equipment behavior and/or the values of other Parameters, but is not considered a control Parameter (for example, E30 "equipment constants" used for configuring equipment communication timeouts, etc.). This class defines no attributes or associations.

10.4.11.9.1 If other Parameters are affected by this Parameter's settings, all such affected Parameters shall have a corresponding AssociatedParameter instance provided with the Configuration Parameter's definition. Each such AssociatedParameter instance shall be accessible via the "affectedParameters" association of this class (inherited from ReadWrite).

10.4.11.9.2 If non-transient, the values of this Parameter are accessible and meaningful at all times, and can be set by an external entity under conditions determined by the equipment.

10.4.11.9.3 If transient, the values of this Parameter are not guaranteed to be meaningful or settable at all times. If the Parameter represents an attribute of an SEMI E39 ObjType, the Parameter will only be settable when the corresponding ObjType has at least one active instance. This class defines no attributes or associations.

10.5 *Parameter Types*

10.5.1 *Parameter Type Definitions*

10.5.1.1 Figure 11 shows a class that can be used to describe re-usable Parameter data types. Parameter type definitions can be re-used elsewhere in the metadata wherever a Parameter's type can be declared. See Related Information Section R5-1 for some examples using the ParameterTypeDefinition class.

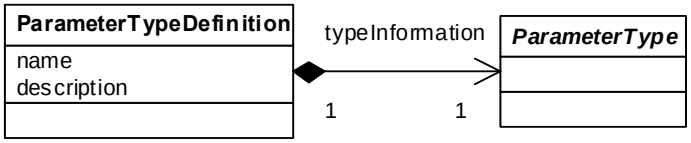

Figure 11
Describing Type Definitions

10.5.1.2 *Parameter TypeDefinition* — Serves as a class for naming and providing a re-usable description of a primitive or composite type. For example, a Parameter TypeDefinition could define a re-usable Parameter type named "KelvinTemperature" of type RealType, with Kelvin units and 4 digits of precision. A composite parameter TypeDefinition can be a named Array, Enumeration, or Structure. Each kind of primitive and composite Parameter type is described fully in subsequent sections of this document. ParameterTypeDefinition can be re-used in the definition of other ParameterTypeDefinitions, recursively within the same ParameterTypeDefinition, and in the definition of Parameters (see Section 10.4).

10.5.1.3 The primitive Parameter type description classes are defined in Section 10.5.2 ; composite type description classes are defined in Section 10.5.3 .

10.5.1.3.1 *ParameterTypeDefinition Attribute Definition Table*

Table 30 ParameterTypeDefinition Attribute Definition

Attribute Name	*Definition*	*Required*	*Form*
name	The name for the re-usable type definition.	Y	Text, must be unique across all ParameterTypeDefinitions.
description	A description of the meaning or purpose of the type definition.	Y	Text.

10.5.1.4 *ParameterTypeDefinition Association Definition Table*

Table 31 ParameterTypeDefinition Association Definition

Association Role Name	*Definition*	*Comments*
typeInformation	The type description.	One and only one element of any type description class derived from ParameterType.

10.5.1.5 *ParameterType* — An abstract base class from which all type description classes are derived. This class has no attributes or associations.

10.5.2 *Primitive Parameter Type Descriptions*

10.5.2.1 Metadata describing types must include a mechanism for describing primitive or single-valued, types such as integers and strings in the type system being used. The specific simple or primitive types that should be provided are typically determined by the implementation technology that will be used to communicate values of the declared type (for example, SECS-II, XML, OMG IDL, etc.). The primitive types described here should be considered a minimum set that could be refined and extended to work best with the type system in use. Any specification that translates these types into a specific type system shall define how it has mapped these primitive types into the technology-specific type system, and shall fully specify any additional primitive type description classes beyond those defined here.

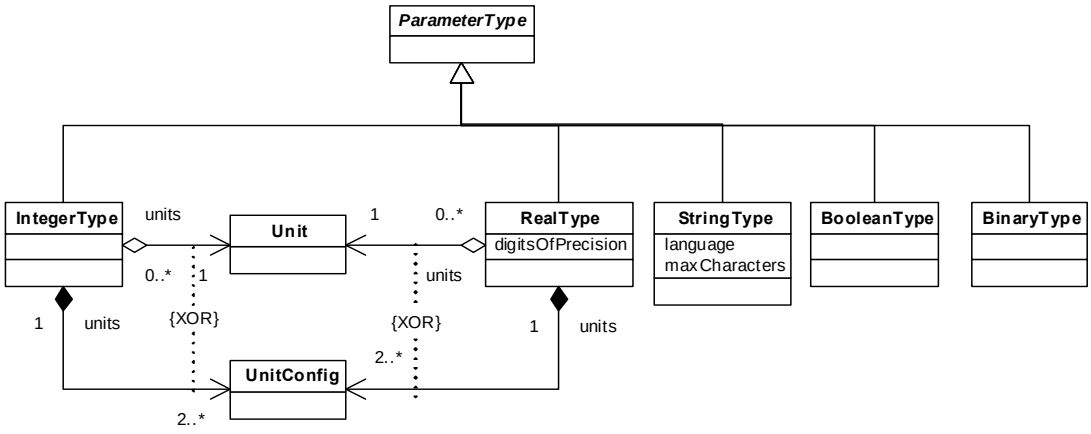

Figure 12
Describing Primitive Parameter Types

10.5.2.2 Figure 12 shows a basic set of classes that can be used to describe primitive Parameter types. The numeric classes representing integer and real numbers can specify their associated units, and real number types can specify the number of digits of precision. Strings can describe the associated language and a maximum number of characters, if applicable.

10.5.2.3 *IntegerType* — Represents an integer type, and associated units.

10.5.2.4 *IntegerType Association Definition Table*

Table 32 IntegerType Association Definition

Association Role Name	*Definition*	*Comments*
units	The available units specified for this type.	Either: One and only one element of type Unit or: Two or more elements of type UnitConfig, if the type supports configurable units.

10.5.2.5 *RealType* — Represents a real (for example, floating point) number type, and associated units.

10.5.2.6 *RealType Attribute Definition Table*

Table 33 RealType Attribute Definition

Attribute Name	*Definition*	*Required*	*Form*
digitsOfPrecision	The maximum number of digits that can be provided after the decimal point by variables of this type.	N	Integer ≥ 0. If 0, there is no restriction on the number of digits of precision.

10.5.2.7 *RealType Association Definition Table*

Table 34 RealType Association Definition

Association Role Name	*Definition*	*Comments*
units	The available units specified for this type.	Either: One and only one element of type Unit or: Two or more elements of type UnitConfig, if the type supports configurable units.

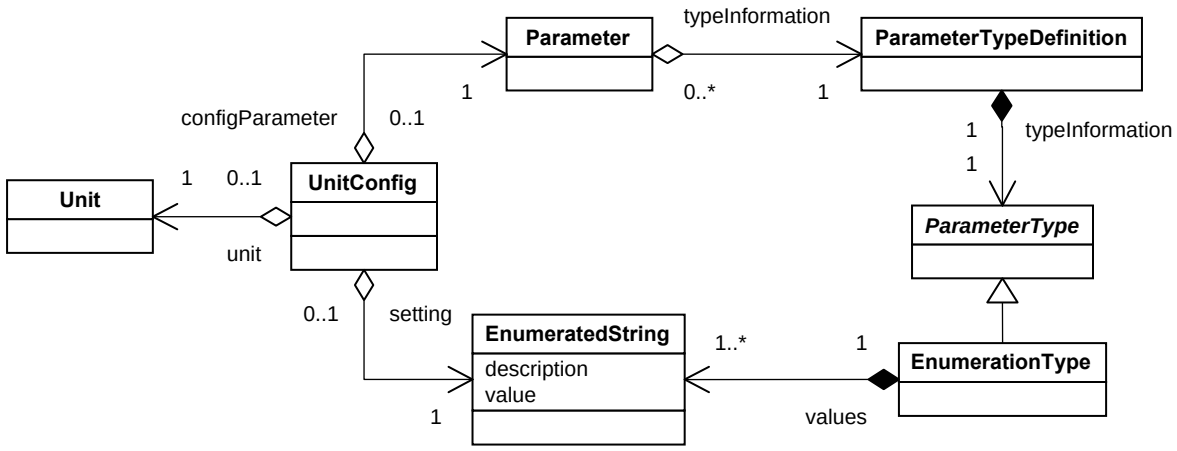

Figure 13
Describing Configurable Units

10.5.2.8 Some equipment may support the ability to provide Parameter values in more than one set of units. For each independently configurable set of units available, the equipment shall support one non-transient unit configuration Parameter to control which units are used when communicating the values of such Parameters. The ParameterType of such unit configuration Parameters shall be a string-based enumeration (see Section 10.5.4.1.3). Each value in the enumeration shall correspond to a different unit configuration. Figure 13 shows the classes used to describe configurable units. Not all associations and attributes of the Unit, Parameter, ParameterTypeDefinition, ParameterType, and EnumerationType classes are shown for clarity.

NOTE 2: The equipment is not required to support configurable units in order to comply with E125; however if the equipment *does* support configurable units, it shall describe the available units and their configuration using the classes defined in this section.

10.5.2.9 The UnitConfig class (shown in Figure 13) is used to provide a mapping from each possible unit choice for a given numeric ParameterType to the enumeration Parameter setting that selects that unit. For example, the equipment may provide a single unit configuration Parameter to select between meter-kilogram-seconds (mks), centimeter-gram-seconds (cgs), and English units. In that case, every numeric type with configurable units appearing in a ParameterTypeDefinition would provide 3 UnitConfig instances – one for each of the 3 possible enumeration settings available from the unit configuration Parameter.

10.5.2.10 Because the mapping from an available unit to the Configuration Parameter setting that selects it is described by the UnitConfig class, it is not required to use the "affectedParameters" attribute (see Section 10.4.11.5.1) to refer to the Parameters whose units are changed by the Configuration Parameter when describing its classification.

10.5.2.11 If a ParameterTypeDefinition includes numeric types with configurable units, then all Parameters that are declared to be of that type shall support the configurable units described in the ParameterTypeDefinition.

10.5.2.12 *UnitConfig* — This class describes a configurable unit setting available for a numeric ParameterType. UnitConfig provides a mapping between the available unit, the non-transient configuration Parameter used to change the selected unit, and the enumeration value of that configuration Parameter that results in the selection of the unit. This class has no attributes.

10.5.2.13 *UnitConfig Association Definition Table*

Table 35 UnitConfig Association Definition

Association Role Name	*Definition*	*Comments*
unit	An available unit supported by this type.	One and only one element of type Unit.
configParameter	The unit configuration Parameter whose value selects the unit.	One and only one element of type Parameter. The Parameter must have a string-based enumeration ParameterTypeDefinition.
setting	The enumeration value which, when set on the corresponding configParameter, results in the available unit being selected.	One and only one element of type EnumeratedString.

10.5.2.14 *StringType* — Represents a string type.

10.5.2.15 *StringType Attribute Definition Table*

Table 36 StringType Attribute Definition

Attribute Name	*Definition*	*Required*	*Form*
language	Identifies the language used to represent the string.	Y	Text, must be equal to an IETF RFC 1766 language tag, using ISO 3166 to name countries and subdivisions, and ISO 639 to name languages.
maxCharacters	The maximum number of characters that this string can contain.	N	Integer ≥ 0. If 0, there is no restriction on the number of characters.

10.5.2.16 *BooleanType* — Represents a Boolean type.

10.5.2.17 *BinaryType* — Represents a binary type.

10.5.3 *Composite Type Description*

10.5.3.1 *Describing Arrays*

10.5.3.1.1 Arrays are container types that can hold one or more values that are of the same type. Figure 14 shows how to describe the type of the elements in a given array. An array can refer to only one of the primitive or composite types, and can declare that it has a maximum number of elements. For example, an array description that has a maxElements value of 0 and refers to the RealType primitive type represents an array of real numbers (for example, float or double) that has no limitation on the number of values it can contain.

10.5.3.1.2 Each element of an array must represent the same physical concept. For example, an array of real numbers must not be used to represent a repeating sequence of a measured value in odd-numbered elements and the uncertainty for the measurement in even-numbered elements. In such cases, an array of StructureType elements is the correct representation, where the different concepts appear as different fields in the structure (for example, a measurement field and an uncertainty field). A human-readable description of the meaning of the array elements is required.

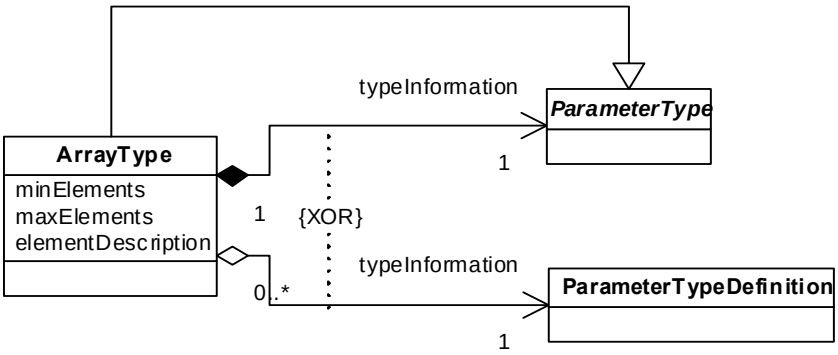

Figure 14
Describing Arrays

10.5.3.1.3 *ArrayType Attribute Definition Table*

Table 37 ArrayType Attribute Definition

Attribute Name	*Definition*	*Form*
minElements	The minimum number of elements that can be held in the array.	Positive integer, with 0 representing no restriction on the minimum number of elements. If this attribute and maxElements are both non-zero, the value of this attribute must be less than maxElements.
maxElements	The maximum number of elements that can be held in the array.	Positive integer, with 0 representing an unlimited number of elements. If this attribute and minElements are both non-zero, the value of this attribute must be greater than minElements.
elementDescription	A human-readable description of the purpose of the elements in the array.	Text.

10.5.3.1.4 *ArrayType Association Definition Table*

Table 38 ArrayType Association Definition

Association Role Name	*Definition*	*Comments*
typeInformation	Describes the data type of each element contained in the array.	Structured data, must be one and only one of the following: any type derived from ParameterType, or a ParameterTypeDefinition (can be a recursive reference from within the same ParameterTypeDefinition).

10.5.3.2 *Describing Structures*

10.5.3.2.1 Structures are composite types that can hold one or more fields that are of differing primitive or composite types. Figure 15 shows how a structure is described as a collection of one or more typed field's.

10.5.3.2.2 *StructureType Attribute Definition Table*

Table 39 StructureType Attribute Definition

Attribute Name	*Definition*	*Form*
description	A human-readable description of the meaning or purpose of the structure.	Text.

10.5.3.2.3 *StructureType Association Definition Table*

Table 40 StructureType Association Definition

Association Role Name	*Definition*	*Comments*
fields	The fields that comprise this data structure.	Ordered list of one or more structured data elements of type Field.

10.5.3.2.4 *Field* — Represents a named field that has one and only one type, primitive or composite. This class is used for describing fields within structures. See Section 10.5.1.2 for a description of the ParameterTypeDefinition class.

10.5.3.2.4.1 *Field Attribute Definition Table*

Table 41 Field Attribute Definition

Attribute Name	*Definition*	*Required*	*Form*
name	The name of the field.	Y	Text.
description	A description of the meaning or purpose of the field.	Y	Text.
canBeNull	Whether or not the field's value may be omitted.		Boolean. True if the field's value is not always provided for all values of the structure.

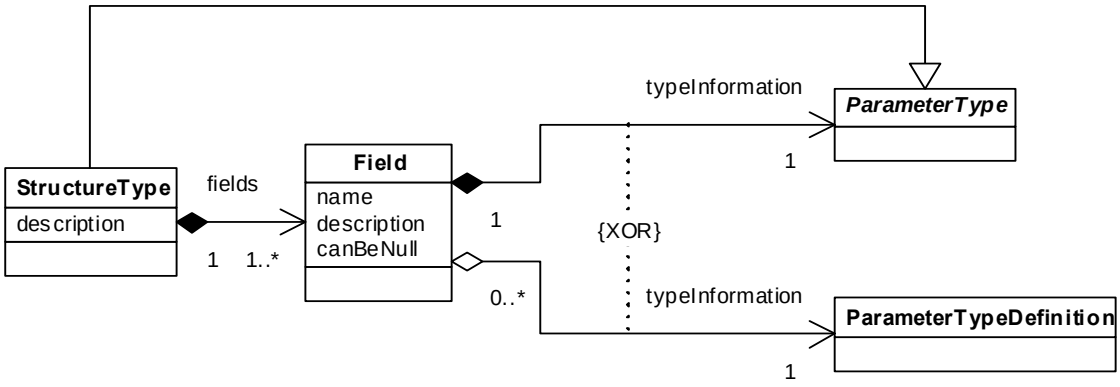

Figure 15
Describing a Structure

10.5.3.2.4.2 *Field Association Definition Table*

Table 42 Field Association Definition

Association Role Name	*Definition*	*Comments*
typeInformation	Describes the data type of this field.	Structured data, must be one and only one of the following: any type derived from ParameterType, or a ParameterTypeDefinition (can be a recursive reference from within the same ParameterTypeDefinition).

10.5.3.3 *Describing Enumerations*

10.5.3.3.1 Enumerations are single-valued types whose possible values are restricted to one of a set of pre-defined values. Figure 16 shows how to describe the valid settings for an enumeration as an ordered collection of integer or string values. For both integer and string enumerations, each value in the enumeration represents a legal value for the type being described.

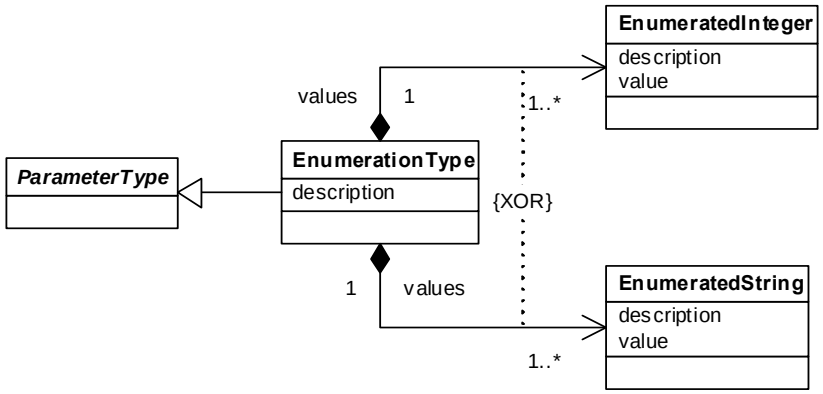

Figure 16
Describing an Enumeration

10.5.3.3.2 *EnumerationType* — Represents an enumeration. EnumerationType may consist of either a set of integers, described by the EnumeratedInteger class, or a set of strings, described by the EnumeratedString class.

10.5.3.3.3 *EnumerationType Attribute Definition Table*

Table 43 EnumerationType Attribute Definition

Attribute Name	*Definition*	*Form*
description	A description of the meaning or purpose of the enumeration as a whole.	Text.

10.5.3.3.4 *EnumerationType Association Definition Table*

Table 44 EnumerationType Association Definition

Association Role Name	*Definition*	*Comments*
values	The set of values that comprise this enumeration.	One of the following: ordered list of one or more elements of type EnumeratedInteger ordered list of one or more elements of type EnumeratedString

10.5.3.3.5 *EnumeratedInteger* — Represents a legal value for an integer enumeration.

10.5.3.3.6 *EnumeratedInteger Attribute Definition Table*

Table 45 EnumeratedInteger Attribute Definition

Attribute Name	*Definition*	*Required*	*Form*
value	A legal value for the enumeration.	Y	Integer, must be unique across all values for this enumeration.
description	A description of the meaning or purpose of this enumeration value.	Y	Text.

10.5.3.3.7 *EnumeratedString* — Represents a legal value for a string enumeration.

10.5.3.3.8 *EnumeratedString Attribute Definition Table*

Table 46 EnumeratedString Attribute Definition

Attribute Name	*Definition*	*Required*	*Form*
value	A legal value for the enumeration.	Y	Text, must be unique across all values for this enumeration.
description	A description of the meaning or purpose of this enumeration value.	Y	Text.

10.5.4 *SEMI Standard Type Mapping*

10.5.4.1 The following table describes how to map SEMI type terminology to the various primitive and composite types defined in this document. This convention shall be followed when describing types that represent types defined in a SEMI standard that is described by metadata.

Table 47 SEMI Type Mapping

SEMI Type Name	*Type Description to use*	*Comments*
integer, positive integer, negative integer, non-negative integer, non-positive integer	IntegerType	
float, floating point, real	RealType	
binary, bit string, byte	BinaryType	
boolean	BooleanType	
text	StringType	If a limitation on the number of characters is specified, the maxCharacters attribute of the StringType class shall be set equal to the number of characters that is supported by the supplier.
enumerated	EnumeratedInteger	Each enumeration shall start at value 0, and is sequenced in increments of 1 according to the order in which the enumerated values are listed in the corresponding standard. The description field for each enumerated value shall be equal to the text name provided for the enumerated value in the corresponding standard.
structure	StructureType	Each field within the structure shall be named exactly as written in the corresponding standard. The type of each field shall be mapped using the conventions described in this table.
list	ArrayType	Lists of integers, floats, binaries, enumerateds, or structures are described as arrays of the corresponding primitives or composite types.

10.5.5 *Parameter Values*

10.5.5.1 Any equipment interface used in conjunction with this specification must provide a means to communicate Parameter values corresponding to the primitive and composite type definitions described in this section. Since this representation will depend on the technology used, the technology-specific specifications for such systems shall provide a mapping between the type descriptions supported by this specification and the representation of their corresponding values.

10.6 *Units*

10.6.1 Figure 17 shows the Unit class that describes a specific unit used by the equipment. Any unit that can be used for numeric data can be described with this class. This specification does not define a system of units, though suppliers shall describe at least one unit that represents pure numbers (a "unit-less" unit).

Figure 17
Describing Units

10.6.2 *Unit Attribute Definition Table*

Table 48 Unit Attribute Definition

Attribute Name	*Definition*	*Required*	*Form*
id	The identifier for this unit.	Y	Text, unique across all units defined for this equipment.
name	The human-readable name for this unit.	Y	Text.
description	A description of the unit, its dimensionality and/or conversion factors, etc.	Y	Text.
symbol	The abbreviation used to represent the unit.	N	Text.

10.7 *Exceptions*

10.7.1 Figure 18 shows how to describe implemented exceptions and their associated data. The attributes and associations of the Parameter class are not shown for clarity (see Section 10.4 for more information about Parameters).

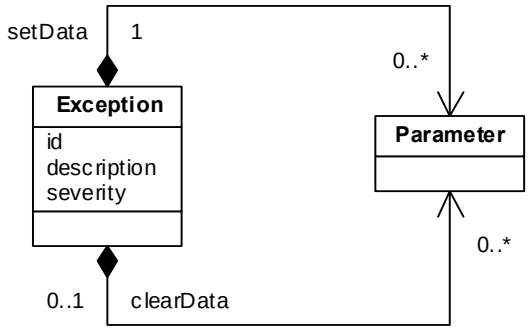

Figure 18
Describing Exceptions

10.7.2 *Exception* — This class can be used to represent a warning, alarm, error, or other abnormal condition that can be communicated by the equipment. Note that the Exception class can represent abnormal conditions whose set/clear states are not tracked by the equipment, as well as those that are (for example, SEMI E30 alarms). For Exceptions whose set/clear states are not tracked, no clearData is provided.

10.7.2.1 *Exception Attribute Definition Table*

Table 49 Exception Attribute Definition

Attribute Name	*Definition*	*Required*	*Form*
id	Unique identifier for this exception.	Y	Text, unique across all exceptions described by metadata.
description	A description of the meaning of the exception when it occurs.	Y	Text.
severity	Identifies the severity of this exception if it occurs.	N	Text, alpha characters only, no spaces. Some examples are: “Informational”, “Warning”, “Error”, or “Fatal”. If no severity is defined, this attribute shall be an empty string.

10.7.3 *Exception Association Definition Table*

Table 50 Exception Association Definition

Association Role Name	*Definition*	*Comments*
setData	An ordered list of data items communicated with the exception when it is detected.	Ordered list of zero or more elements of type Parameter (see Section 10.4).
clearData	An ordered list of data items communicated when the exception has been cleared.	Ordered list of zero or more elements of type Parameter (see Section 10.4).

10.8 *StateMachines*

10.8.1 *State Machine Descriptions*

10.8.1.1 Figure 19 shows how to describe the definition of a StateMachine (and its associated states and transitions) that is implemented by the equipment. These classes can be used to describe standardized state machines, such as those originating from SEMI standards, as well as non-standardized equipment-specific state machines. Examples of state machines are provided in Related Information, Section R7-1.

10.8.1.1.1 The state machine description classes specified here are intended for descriptive purposes in facilitating end-user understanding of equipment behavior that results in the generation of events. They are not intended to provide comprehensive modeling of all semantics defined by standard state machine formalisms such as Harel or UML. Events generated by the equipment must be defined within the context of these state machine descriptions, even if the described state machine has not been literally implemented by the supplier. Further, this specification does not require the supplier to provide a literal implementation of the state machines described by these classes.

10.8.1.2 A StateMachine that defines behavior that is implemented by many equipment components (for example) need only be defined once. Each component that implements the behavior described by that StateMachine can then refer to the events it defines through the use of the StateMachineInstance class (see Section 10.8.6). It is not necessary to define multiple StateMachines to accommodate multiple components that each exhibit the same behavior and generate the same events. For example, if the equipment being described has four pump components that exhibit the same behavior, have the same underlying StateMachine, and generate the same set of events, only one StateMachine description is necessary. Each pump component can then act as a different source (see Section 10.3.1) of the same set of events defined for that StateMachine, each pump executing the StateMachine independently of the other pumps.

10.8.2 *StateMachine* — This class represents a collection of states and transitions that together comprise a non-concurrent finite state machine. All StateMachines refer to a single top-level state. The top-level state can contain zero or more substates and/or StateMachines.

10.8.2.1 The top-level StateMachine instance in a complete state model description represents the state model as a whole. When appearing as a member of the 'stateMachines' association of the State class, the StateMachine class can be used to describe 2 or more concurrent state machines belonging to that State instance. For example, the SEMI E87 Carrier state model defines three such concurrent state machines: Carrier ID Status, Carrier Slot Map Status, and Carrier Accessing Status.

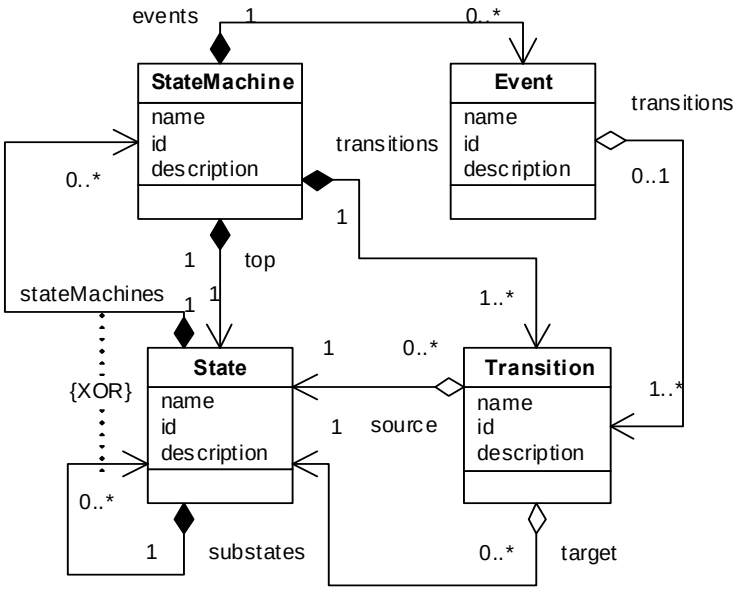

Figure 19
Describing State Machines

10.8.2.2 *StateMachine Attribute Definition Table*

Table 51 StateMachine Attribute Definition

Attribute Name	*Definition*	*Required*	*Form*
name	A human-readable name for this StateMachine.	Y	Text.
id	A unique identifier for this StateMachine.	Y	Text, unique across all state machines.
description	A description of the meaning or purpose of this state model.	Y	Text.

10.8.2.3 *SEMI Standard State Machines*

10.8.2.3.1 *name* — If the StateMachine represents a SEMI standard state machine, the value of this attribute shall be a URN of the form: "urn:semi-org:stateMachine:<standard id>:<state machine>".

10.8.2.3.1.1 <standard id> shall be a string formatted according to the SEMI standard form for representing a specific version of a standard document (for example, "E87-1101"), no spaces, upper-case alphanumeric characters and hyphens only.

10.8.2.3.1.2 <state machine> shall be the name of the state machine as it appears in the standard, with each word in the name concatenated. The state machine name shall use the camel case capitalization convention (that is, only the first letter of each word in the state machine name is capitalized). For example, the state machine name for the SEMI E87 LOAD PORT TRANSFER state machine would be "LoadPortTransfer".

10.8.2.4 *StateMachine Association Definition Table*

Table 52 StateMachine Association Definition

Association Role Name	*Definition*	*Comments*
top	The top-level state for this state model.	One and only one element of type State.
transitions	A list of zero or more transitions between the states and substates that comprise this state machine.	List of one or more elements of type Transition.
events	The events that the equipment generates when a transition between states occurs	List of zero or more elements of type Event.

10.8.3 *State* — This class represents a single state contained within a state machine. Each State can contain zero or more substates, or zero or more StateMachines, but cannot contain both substates and StateMachines. A substate represents a state-within-a-state. A StateMachine represents a state-machine-within-a-state. For example, if a given state contained 2 concurrently executing state machines, each concurrent state machine would be represented as a stateMachine within the containing state. Initial states shall have a name attribute equal to "Initial", final states shall have a name attribute equal to "Final".

10.8.3.1 *State Attribute Definition Table*

Table 53 State Attribute Definition

Attribute Name	*Definition*	*Required*	*Form*
name	A human-readable name for this state.	Y	Text.
id	A unique identifier for this state.	Y	Text, unique across all states in this StateMachine.
description	A description of the meaning or purpose of this state.	Y	Text.

10.8.3.2 *SEMI Standard States*

10.8.3.2.1 *name* — If the State represents a SEMI standard state, the value of this attribute shall be a URN of the form: "urn:semi-org:state:<standard id>:<state machine>:<state path name>".

10.8.3.2.1.1 <standard id> shall be a string formatted according to the SEMI standard form for representing a specific version of a standard document (for example, "E87-1101"), no spaces, upper-case alphanumeric characters and hyphens only.

10.8.3.2.1.2 <state machine> shall be the name of the state machine in which the transition is defined. This field shall be equal to the state machine name as it appears in the standard, with each word in the name concatenated. The state machine name shall use the camel case capitalization convention (that is, only the first letter of each word in the state machine name is capitalized). For example, the state machine name for the SEMI E87 LOAD PORT TRANSFER state machine would be "LoadPortTransfer".

10.8.3.2.1.3 <state path name> shall be formed by concatenating the names of all states from the top level state in the state machine to the state being described, including any substates in between. Each state name in the path shall be separated by the '.' character. State names shall have no spaces, use alphanumeric characters only, and use camel case capitalization. For example, the E87 "SLOT MAP NOT READ" state of the Carrier state machine would have a <state path name> equal to "Carrier.CarrierSlotMapStatus.SlotMapNotRead".

10.8.3.3 *State Association Definition Table*

Table 54 State Association Definition

Association Role Name	*Definition*	*Comments*
substates	A list of zero or more substates that comprise this state.	List of zero or more elements of type State.
stateMachines	A list of zero or more concurrently-executing state models that comprise this state.	List of zero or more elements of type StateMachine.

10.8.4 *Transition* — This class represents a transition between any two states contained within the same *StateMachine,* including substates. Each transition references one "source", or "from" state, and one "target", or "to" state, representing the originating and ending states. There must be one Transition defined for all possible state transitions in the *StateMachine* instance. History or conditional transitions are described by providing as many transitions as there are possible conditional or history-based outcomes. True Harel history transitions can be accommodated by mapping all such transitions to a single event. The 'description' attribute can be used to provide a human-readable description of the condition under which the transition takes place.

10.8.4.1 *Transition Attribute Definition Table*

Table 55 Transition Attribute Definition

Attribute Name	*Definition*	*Required*	*Form*
id	A unique identifier for this transition.	Y	Text, unique across all transitions in this StateMachine.
name	A human-readable name for this transition.	N	Text.
description	A description of the meaning or purpose of this transition.	Y	Text.

10.8.4.2 *SEMI Standard Transitions*

10.8.4.2.1 *name* — If the Transition represents a SEMI standard transition, the value of this attribute shall be a URN of the form: "urn:semi-org:transition:<standard id>:<state machine>:<transition number>:<transition name>".

10.8.4.2.1.1 <standard id> shall be a string formatted according to the SEMI standard form for representing a specific version of a standard document (for example, "E87-1101"), no spaces, upper-case alphanumeric characters and hyphens only.

10.8.4.2.1.2 <state machine> shall be the name of the state machine in which the transition is defined. This field shall be equal to the state machine name as it appears in the standard, with each word in the name concatenated. The state machine name shall use the camel case capitalization convention (that is, only the first letter of each word in the state machine name is capitalized). For example, the state machine name for the SEMI E87 LOAD PORT TRANSFER state machine would be "LoadPortTransfer".

10.8.4.2.1.3 <transition number> shall be a string representation of the transition number as it appears in the state machine definition.

10.8.4.2.1.4 <transition name> shall be formed by concatenating the names of the source and target states for the <transition number>, separated by a hyphen. The state names shall have no spaces, use alphanumeric characters only, and use camel case capitalization. If either of these states is an "initial" or "final" state, the words "Initial" or "Final", respectively, shall be used for the state name. For example, the event name for the E87 load port transfer history transition from the initial state to "OUT OF SERVICE" would be "Initial-OutOfService".

10.8.4.3 *Transition Association Definition Table*

Table 56 Transition Association Definition

Association Role Name	*Definition*	*Comments*
source	The state of origin for this transition.	References one and only one state within this StateMachine.
target	The transitioned-to state.	References one and only one state within this StateMachine.

10.8.5 *Event* — This class defines an event that the equipment generates as a result of a transition between any two states contained within the same *StateMachine*, including substates. Each Event can reference one or more transitions. This is in support of systems or standards that use the same event id to communicate different transitions. Such implementations typically make additional data available with the event occurrence for distinguishing between the different transitions. Note that it is not possible for two different events to be defined for the same transition.

10.8.6 *State Machine Instances*

10.8.6.1 State machines are intended to be defined once and re-used wherever there is an equipment component or other entity that can generate the same set of events defined by that state machine. However, each entity that supports a given state machine may report different data for those events. Figure 20 shows the classes that are used to describe the Parameters available for reporting with the events from a given state machine supported by a given entity.

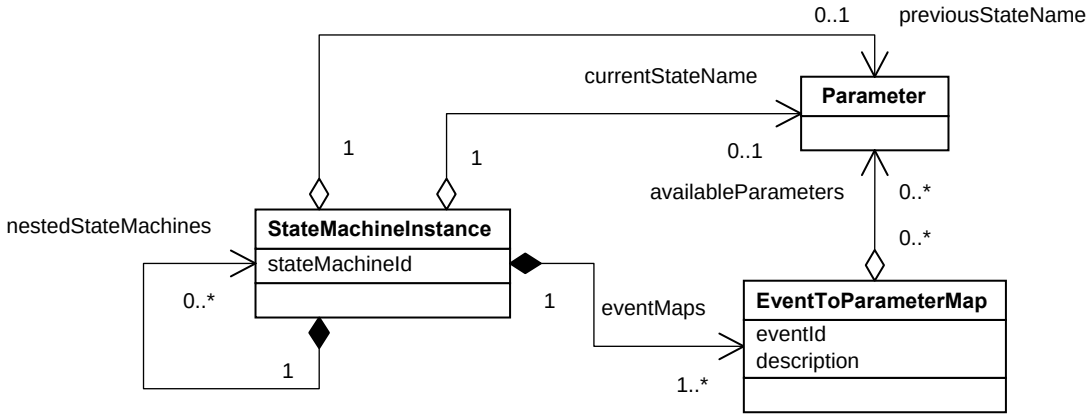

Figure 20
Describing State Machine Instances

10.8.6.2 *StateMachineInstance* — This class is used to declare that an equipment node or other entity supports a specific state machine described in the metadata. It provides a way to describe the data such entities provide with the events from that state machine as well as the Parameters used to determine the current state of the entity that supports the state machine. The class can be used to describe the events and data available from top-level state machines as well as any nested state machines defined within the top level state machine.

10.8.6.3 *StateMachineInstance Attribute Definition Table*

Table 57 StateMachineInstance Attribute Definition

Attribute Name	*Definition*	*Form*
stateMachineId	Identifies a top level or nested state machine.	Text, equal to a valid 'id' attribute of the StateMachine class.

10.8.6.4 *StateMachineInstance Association Definition Table*

Table 58 StateMachineInstance Association Definition

Association Role Name	*Definition*	*Comments*
currentStateName	The Parameter, if any, that can be accessed to determine the current state within the identified StateMachine.	Zero or one element of type Parameter, described in Section 10.4 .
previousStateName	The Parameter, if any, that can be accessed to determine the previous state within the identified StateMachine.	Zero or one element of type Parameter, described in Section 10.4 .
eventMaps	The events defined by this StateMachine, and the Parameters that are available for reporting with those events.	One or more elements of type EventToParameterMap, described in Section 10.8.6.5 . There shall be one such map for each event defined in the identified StateMachine.
nestedStateMachines	Any nested state machines defined within the identified StateMachine.	One or more elements of type StateMachineInstance. Although it is possible to create more, only one level of nesting of the StateMachineInstance class is required to describe arbitrary levels of nesting described by the referenced top-level StateMachine.

10.8.6.5 *EventToParameterMap* — This class is used to provide a list of the Parameters available for a specific event from a given StateMachine.

10.8.6.6 *EventToParameterMap Attribute Definition Table*

Table 59 EventToParameterMap Attribute Definition

Attribute Name	*Definition*	*Form*
eventId	The StateMachine Event for which Parameters are available	Text, must be equal to a valid 'id' attribute of the Event class within a StateMachine.
description	Provides any description necessary for humans to understand the parameters that are available for reporting with the event.	Text.

10.8.6.7 *EventToParameterMap Association Definition Table*

Table 60 EventToParameterMap Association Definition

Association Role Name	*Definition*	*Comments*
availableParameters	The Parameters that are available for reporting with this event.	Structured data of type Parameter, described in Section 10.4.

10.8.6.7.1 *availableParameters* — All transient Parameters (see Section 10.4.11) that are available for reporting with an event generated by the entity providing the EventToParameter map shall be provided in this list. It is not necessary to list any non-transient Parameters via this association. Non-transient Parameters are implicitly available for reporting with any event.

10.8.6.8 Events may only be defined within the context of a Transition between two States within a StateMachine. All events that the equipment can generate must have a StateMachine defined for them. It is not necessary to define multiple StateMachines for a situation in which the same kind of events are generated by multiple sources.

10.8.6.9 *Event Attribute Definition Table*

Table 61 Event Attribute Definition

Attribute Name	*Definition*	*Form*
name	A human-readable name for the event.	Text.
id	A unique identifier for the event.	Text, unique across all events in all StateMachines.
description	A human-readable description of the meaning or purpose of the event.	Text.

10.8.6.10 *SEMI Standard Events*

10.8.6.10.1 *name* — If the Event represents a SEMI standard event, the value of this attribute shall be a URN of the form: "urn:semi-org:event:<standard id>:<state machine>:<transition number>:<event name>".

10.8.6.10.1.1 <standard id> shall be a string formatted according to the SEMI standard form for representing a specific version of a standard document (for example, "E87-1101"), no spaces, upper-case alphanumeric characters and hyphens only.

10.8.6.10.1.2 <state machine> shall be the name of the state machine in which the event is defined. If no such state machine is defined, this field shall be equal to the text "None". If there is a defined state machine, the name shall be equal to the state machine name as it appears in the standard, with each word in the name concatenated. The state machine name shall use the camel case capitalization convention (that is, only the first letter of each word in the state machine name is capitalized). For example, the state machine name for the SEMI E87 LOAD PORT TRANSFER state machine would be "LoadPortTransfer".

10.8.6.10.1.3 <transition number> shall be a string representation of the transition number as it appears in the state machine definition. If no state machine or transition number is defined in the standard, this field shall be equal to the text "None".

10.8.6.10.1.4 <event name> shall be the name of the event as it appears in the standard with no spaces, alphanumeric characters only, using camel case capitalization. If no event name is defined, this field shall be formed by concatenating the names of the source and target states for the <transition id>, separated by a hyphen. The state names shall have no spaces, alphanumeric characters only, and use camel case capitalization. If either of these states is an "initial" or "final" state, the words "Initial" or "Final", respectively, shall be used for the state name. For example, the event name for the E87 load port transfer history transition from the initial state to "OUT OF SERVICE" would be "Initial-OutOfService".

10.8.6.11 *Event Association Definition Table*

Table 62 Event Association Definition

Association Role Name	*Definition*	*Comments*
transitions	The transitions to which this event is mapped.	One or more elements of type Transition.

10.9 *ObjTypes*

10.9.1 Figure 21 shows how to describe SEMI ObjTypes (for example, the "Carrier" ObjType from SEMI E87) that have been implemented by the equipment. An ObjType is similar to a class in that it describes attributes and state machines that are common to all instances of the ObjType. Each ObjType instance maintains its attributes and state machines independently of all other instances of the same ObjType. See Related Information 6 for examples using the SEMIObjType class.

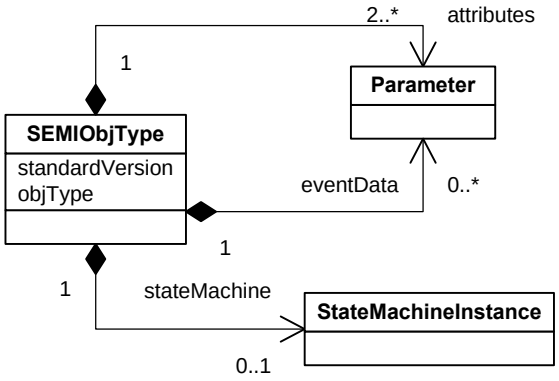

Figure 21
Describing Implemented SEMI ObjTypes

10.9.1.1 *SEMIObjType* — This class describes the attributes and events for a SEMI ObjType that has been implemented by the equipment. If it is a SEMI standard ObjType then it is identified by the name of the SEMI standard within which it was defined, and the name of the ObjType specified in that standard. Each ObjType includes at least two attributes, corresponding to its type name and its instance id (see SEMI E39).

10.9.1.2 *SEMIObjType Attribute Definition Table*

Table 63 SEMIObjType Attribute Definition

Attribute Name	*Definition*	*Required*	*Form*
standardVersion	The complete SEMI standard version number for the document where the ObjType is defined, if applicable.	Y	Text, using the SEMI standard form for representing a specific version of a standard document (for example, "E87-1101"), no spaces, upper-case alphanumeric characters and hyphens only. If not a SEMI standard ObjType, then this attribute shall be empty.
objType	The name of the SEMI objType.	Y	Text. Named using identical spelling and case as the documented objType in the corresponding standard.

10.9.1.3 *SEMIObjType Association Definition Table*

Table 64 SEMIObjType Association Definition

Association Role Name	*Definition*	*Comments*
attributes	Attributes defined for this ObjType.	Unordered list of elements of type Parameter, described in Section 10.4 . Each Parameter shall be typed and named according to Sections 10.5.4 and 10.4.10 . All attributes belonging to an ObjType are maintained independently in each ObjType instance.
eventData	Parameters that are not formal attributes that this ObjType provides with generated events.	List of zero or more elements of type Parameter, described in Section 10.4 .
stateMachine	The state machine (if any) supported by this ObjType.	Zero or one element of type StateMachineInstance, described in Section 10.8.6 .

11 Requirements for Compliance

11.1 *Compliance to Equipment Self-Description*

11.2 Table 55 provides a checklist for Equipment Self Description compliance.

Table 65 Equipment Self-Description Compliance Statement

Fundamental Requirements	*Section*	*Implemented using SEMI technology mapping*	*Implementation complies with ESD specification and technology mapping*	*ESD data associated with all information sources and items accessible via data acquisition are described*
Metadata Access	9.1	□ Yes □ No	□ Yes □ No	N/A
Management of Equipment Metadata	10.1	□ Yes □ No	□ Yes □ No	N/A
Equipment Physical Structure	10.2	□ Yes □ No	□ Yes □ No	□ Yes □ No
Equipment Node Description	10.3	□ Yes □ No	□ Yes □ No	□ Yes □ No
Parameters	10.4	□ Yes □ No	□ Yes □ No	□ Yes □ No
Types	10.5	□ Yes □ No	□ Yes □ No	□ Yes □ No
Units	10.6	□ Yes □ No	□ Yes □ No	□ Yes □ No
Exceptions	10.7	□ Yes □ No	□ Yes □ No	□ Yes □ No
State Machines	10.8	□ Yes □ No	□ Yes □ No	□ Yes □ No
ObjTypes	10.9	□ Yes □ No	□ Yes □ No	□ Yes □ No

RELATED INFORMATION 1
TYPICAL USE CASES

NOTICE: This related information is not an official part of SEMI E125 and was derived from the work of the originating committee. This related information was approved for publication by full letter ballot procedures.

R1-1 Example Use Cases

R1-1.1 Figure R1-1 shows an example of some typical use cases that this specification supports. The actors shown represent some of the possible applications that could make use of the information provided by this specification, and are only examples. In an actual factory, other applications/actors not shown here may make use of equipment metadata. The system boundary in this diagram represents all entities that comprise or represent a single equipment installation.

R1-2 Example Actors

R1-2.1 *EquipmentDiagnosticsClient* — This actor represents an application that helps factory or supplier personnel troubleshoot mechanical, electrical, chemical, or other problems identified with the equipment.

R1-2.2 *UtilizationTrackingClient* — This actor represents an application that collects performance information from the equipment to calculate relevant utilization metrics.

R1-2.3 *ProcessControlClient* — This actor represents an application that orchestrates material processing, measurement, or testing performed by the equipment.

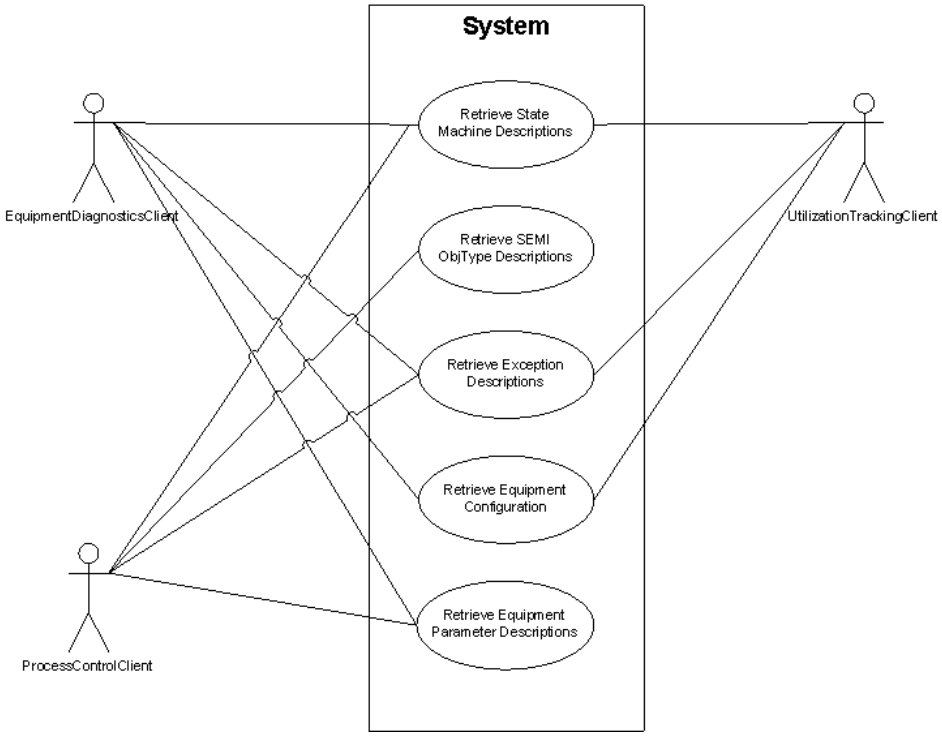

Figure R1-1
Typical Equipment Metadata Use Cases

R1-3 Use Case Descriptions

R1-3.1 *Retrieve State Machine Descriptions* — This use case provides a way for an application to request a description of all of the state machines supported by the equipment, and to determine which state machines are implemented by which components of the equipment. This supports automated usage, so that applications that are looking for standardized or well-known equipment-specific state models can discover their existence on a specific tool, as well as the supplier-specific event id's used to communicate standardized state transitions. It also supports any manual application in which a human user would like to understand equipment-specific behavior that has not been standardized. Figure R1-2 shows one example of how this use case can be realized using messages from the interface defined in this specification (see Section 9).

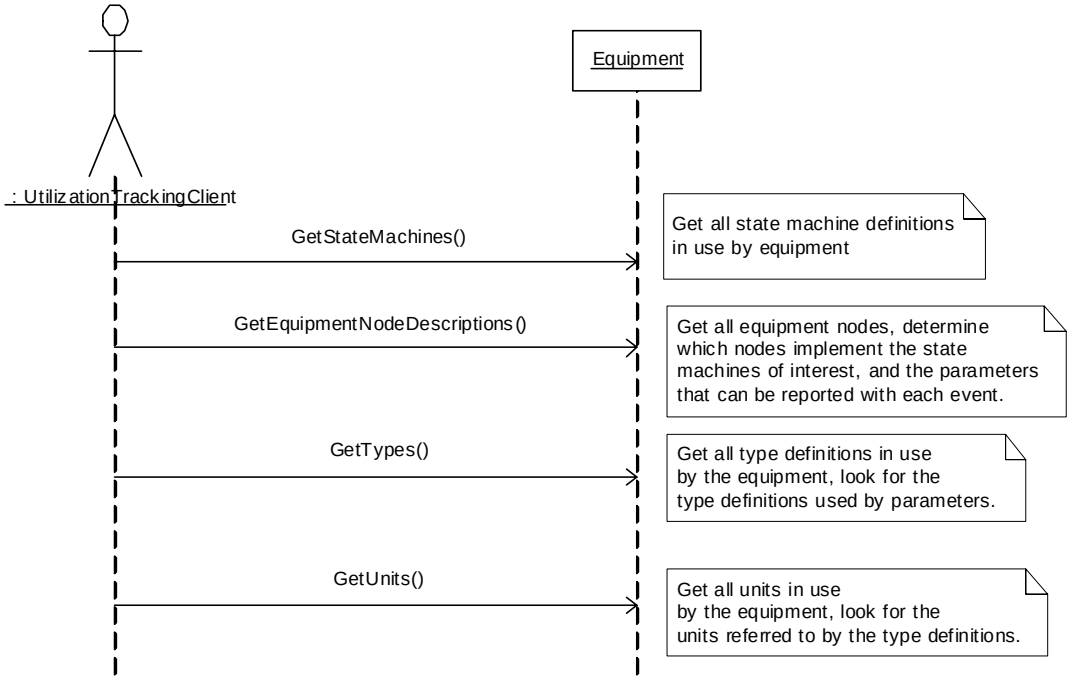

Figure R1-2
Use Case Realization for "Retrieve State Machine Descriptions"

R1-3.2 *Retrieve SEMIObjType Descriptions* — This use case provides a way for an application to request a description of all of the SEMI standard OSS ObjTypes supported by the equipment. This supports automated usage, so that applications that are written to work with OSS ObjTypes can discover which ObjTypes are in use on a specific equipment installation, and can learn the event id's that the supplier uses for standardized state transitions. Figure R1-3 shows one example of how this use case can be realized using messages from the interface defined in this specification (see Section 9).

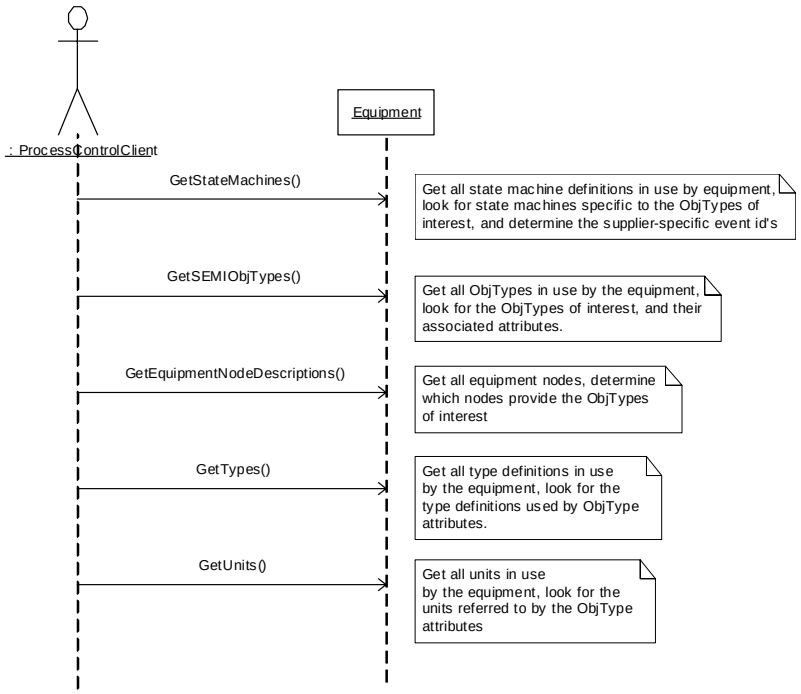

Figure R1-3
Use Case Realization for "Retrieve ObjType Descriptions"

R1-3.3 *Retrieve Exception Descriptions* — This use case provides a way for an application to request a description of all of the exceptions generated by the equipment, and to determine which components of the equipment generate which exceptions. This supports automated usage, so that applications that are looking for standardized or well-known equipment specific exceptions can discover their existence on a specific tool. It also supports any manual usage in which a human user would like to understand available equipment-specific exceptions. Figure R1-4 shows one example of how this use case can be realized using messages from the interface defined in this specification (see Section 9).

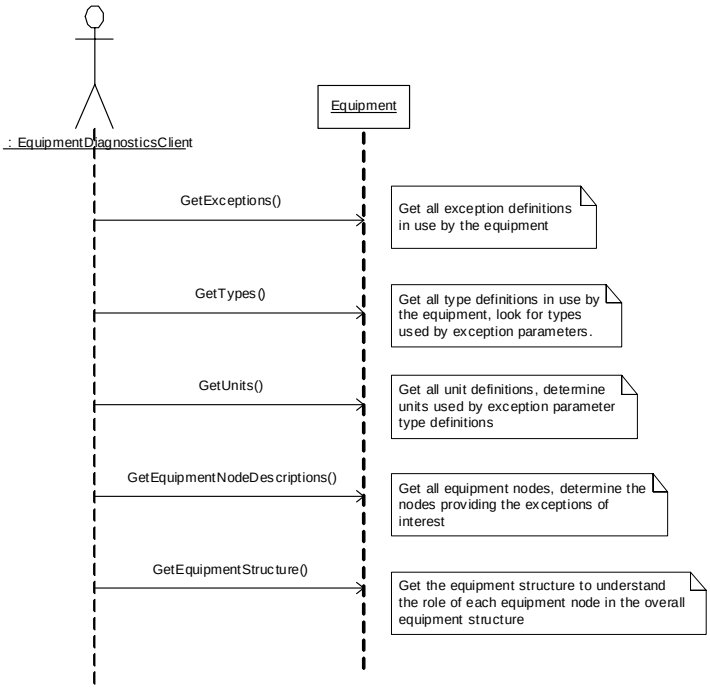

Figure R1-4
Use Case Realization for "Retrieve Exception Descriptions"

R1-3.4 *Retrieve Equipment Parameter Descriptions* — This use case provides a way for an application to request a description of all of the data items provided by the equipment, and to determine which components of the equipment provide that data. This supports automated usage, so that applications that are looking for standardized or well-known equipment-specific data items can discover their existence on a specific tool. It also supports any manual usage in which a human user would like to browse and understand equipment-specific data that has not been standardized. Figure R1-5 shows one example of how this use case can be realized using messages from the interface defined in this specification (see Section 9).

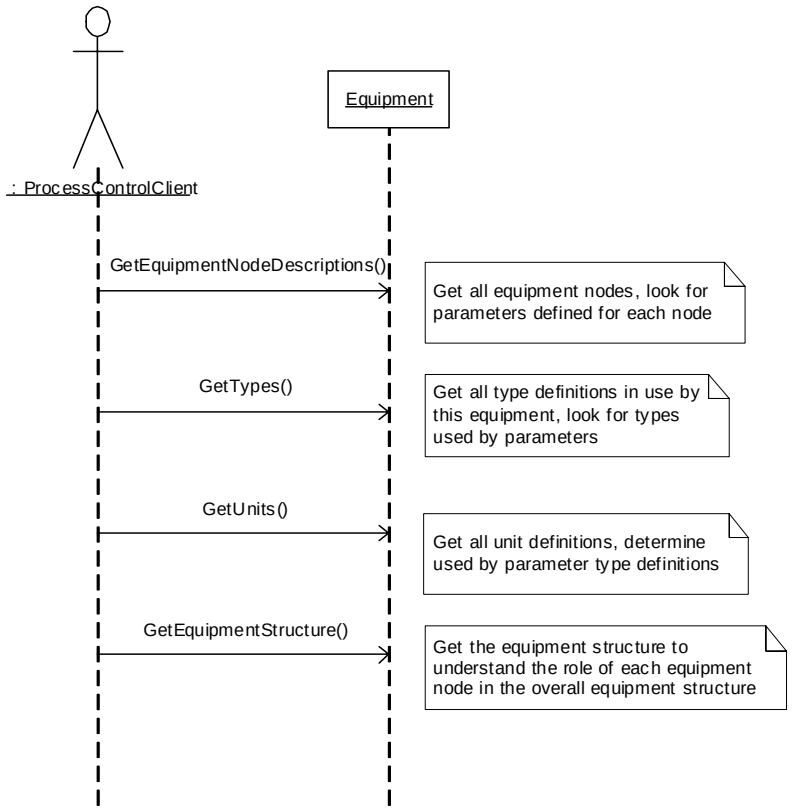

Figure R1-5
Use Case Realization for "Retrieve Equipment Parameter Descriptions"

R1-3.5 *Retrieve Equipment Configuration* — This use case is directly supported by this specification, and provides a way for an application to request a description of the physical and logical equipment configuration. Figure R1-6 shows one example of how this use case can be realized using messages from the interface defined in this specification (see Section 9).

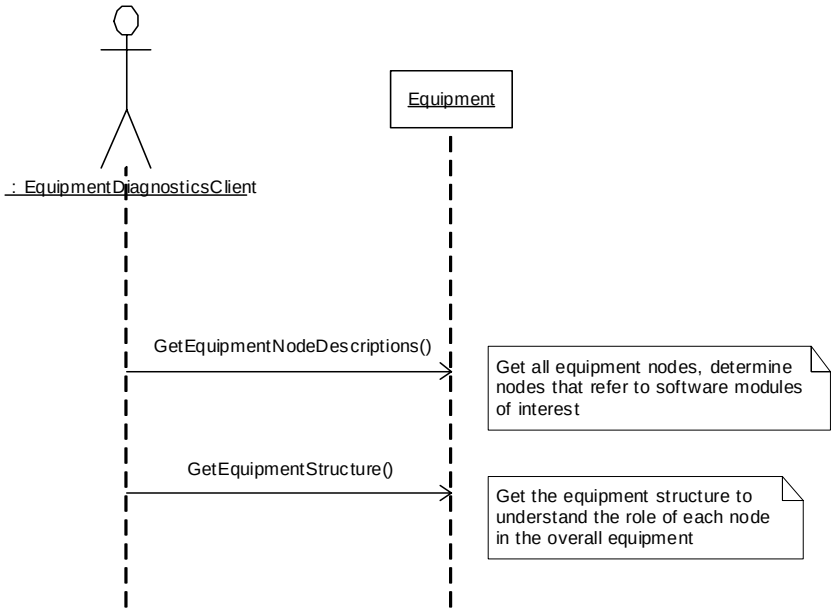

Figure R1-6
Use Case Realization for "Retrieve Equipment Configuration"

RELATED INFORMATION 2
SQL WHERE CLAUSE

NOTICE: This related information is not an official part of SEMI E125 and was derived from the work of the originating committee. This related information was approved for publication by full letter ballot procedures.

R2-1 Syntax

R2-1.1 The **WHERE** clause of the SQL **SELECT** statement is a propositional-expression that consists of one or more comparisons of values with constants. It has the following form.

WHERE *propositional-expression* [*nested-propositional-expression*];

R2-1.2 *Syntax*

R2-1.2.1 *propositional-expression* — Specifies a comparison. It takes one of the following forms:

- [**NOT**] *term operator term*
- *parameter* [**NOT**] **IN** (*constant*$_1$,… *constant*$_n$)
- *parameter* [**NOT**] **BETWEEN** (*term*$_1$ **AND** *term*$_2$) (inclusive)
- *term* **IS** [**NOT**] **NULL** | **TRUE** | **FALSE**
- *parameter* [**NOT**] **LIKE** "*string%*"

R2-1.2.2 [ ] — An optional equation item.

R2-1.2.3 *term* — Specifies a parameter, a constant or an arithmetic expression of parameters and constants. The allowed arithmetic operations are addition (+), subtraction (-), multiplication (*) and division (/).

R2-1.2.4 *parameter* — Specifies the name of the parameter value to use in the comparison operation.

R2-1.2.5 *Constant*

1) Specifies a number or a string enclosed by quotation marks.

2) Specifies a conditional. A conditional has the following form.

DECODE(term**,** constant$_1$**,** return$_1$**,** … constant$_n$**,** return$_n$, default)

return$_i$

The value of the conditional if term equals constant$_i$.

default

The value of the conditional if parameter does not equal any constant$_i$.

R2-1.2.6 *operator* — Specifies the operator to use in the comparison. It can be one of the following operators.

Operator	*Description*
=	Equal
>	Greater than
>=	Greater than or equal to

Operator	*Description*
<>	Not equal
<	Less than
<=	Less than or equal to

R2-1.3 IN — List Inclusion test (enumeration comparison).

R2-1.4 BETWEEN — Bounds test (limits comparison).

R2-1.5 IS — Equality test.

R2-1.6 LIKE — Pattern matching (string comparison) test. (**LIKE** only works with ASCII data types).

R2-1.6.1 *A* | *B* | … *N* — One of the listed items is required.

R2-1.6.2 *String%* — String constant followed by the "%" or "*" wild character. A wildcard character can also be used by itself to match everything. Use a wildcard character only at the end of the string constant.

R2-1.6.3 nested-propositional-expression — AND|OR propositional-formula [nested-propositional-formula]

NOTE 1: A semicolon is required at the end of SQL statements. The ';' indicates the SQL statement is complete.

NOTE 2: Parentheses can be used in the usual way to group expressions and establish precedence.

NOTE 3: Floating-point comparisons are approximate. Testing for equality with floating point values is not recommended.

RELATED INFORMATION 3
PARAMETER CONSTRAINTS

NOTICE: This related information is not an official part of SEMI E125 and was derived from the work of the originating committee. This related information was approved for publication by full letter ballot procedures.

R3-1 Examples

R3-1.1 This clause specifies that all values of "voltage" that are less than or equal to 50 are valid (i.e., it evaluates true for all values of voltage less than or equal to 50).

WHERE voltage <= 50;

R3-1.2 The following clauses specifies that all values of "voltage" that are between 0 and 50 (inclusive) are valid.

WHERE voltage >= 0 **AND** voltage <= 50;

R3-1.3 The clause specifies that all values of "voltage" that are 50 and less or 100 and greater (inclusive) are valid.

WHERE voltage <= 50 **OR** voltage >= 100;

R3-1.4 The following clause specifies that only "voltage" values of 1, 2, 4, 8 and 16 are valid.

WHERE voltage = 1 **OR** voltage = 2 **OR** voltage = 4 **OR** voltage = 8 **OR** voltage = 16;

R3-1.5 The following clause specifies that any "voltage" value that is *not* 1, 2, 4, 8 or 16 is valid.

WHERE voltage <> 1 **AND** voltage <> 2 **AND** voltage <> 4 **AND** voltage <> 8 **AND** voltage <> 16;

RELATED INFORMATION 4
PARAMETER REPORTING PERIOD CONSTRAINTS

NOTICE: This related information is not an official part of SEMI E125 and was derived from the work of the originating committee. This related information was approved for publication by full letter ballot procedures.

R4-1 Examples

R4-1.1 The constraint definition below specifies that the supported reporting period for "voltage" is any continuous value between 0.01 seconds and 60 seconds.

WHERE voltage.ReportingPeriod > .01 **AND** voltage.ReportingPeriod < 60;

R4-1.2 The constraint definition below specifies that the supported reporting period for "voltage" is any integer multiple (from 1 to 6000) of a fundamental period of 0.01 seconds.

WHERE voltage.ReportingPeriod = **(** n*.01**) AND** n > 1 **AND** n < 6000;

R4-1.3 The constraint definition below specifies that the supported reporting period for "voltage" is any of the discrete values 0.05 seconds, 0.07 seconds, 0.1 seconds, or 1.5 seconds.

WHERE voltage.ReportingPeriod = .05 **OR** voltage.ReportingPeriod =.07 **OR** voltage.ReportingPeriod = .1 **OR** voltage.ReportingPeriod = 1.5

RELATED INFORMATION 5
TYPE DEFINITIONS

NOTICE: This related information is not an official part of SEMI E125 and was derived from the work of the originating committee. This related information was approved for publication by full letter ballot procedures.

R5-1 Examples

R5-1.1 Figure R5-1 shows an example set of type definitions using the classes described in Section 10.5. These types could be used for Parameters used to represent the SEMI standard ObjType, ObjId, and ContentMap attributes of the SEMI E87 Carrier ObjType, as well as re-usable type definitions for the strings used for the ObjID and ObjType attributes defined in SEMI E39. Note that the type descriptions conform to the SEMI type mapping described in Section 10.5.7.

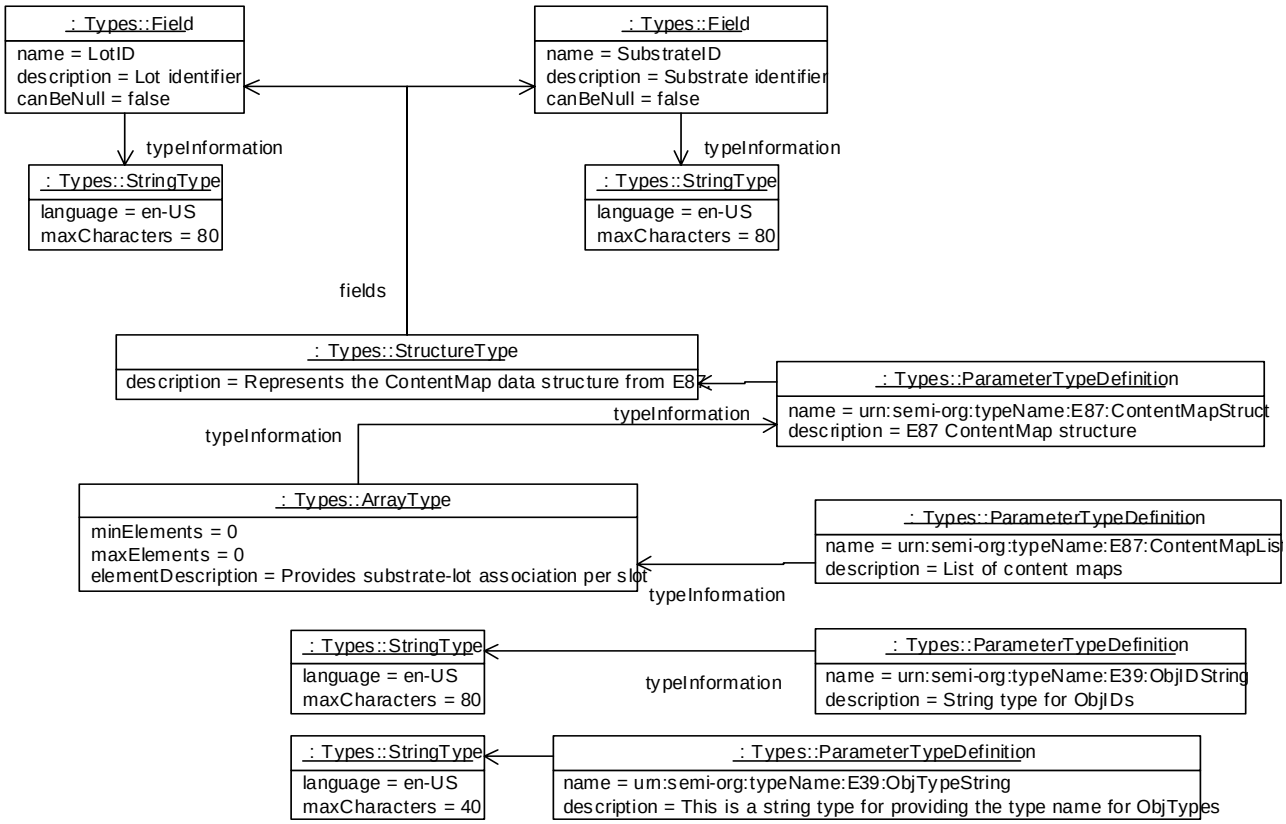

Figure R5-1
TypeDefinition Example

RELATED INFORMATION 6
OBJTYPES

NOTICE: This related information is not an official part of SEMI E125 and was derived from the work of the originating committee. This related information was approved for publication by full letter ballot procedures.

R6-1 Examples

R6-1.1 Figure R6-1 shows an abbreviated example of how to describe the E87 Carrier ObjType using the classes from Section 10.10. Not all events and attributes available for the Carrier ObjType are shown for clarity, only the ObjType, ObjId, and ContentMap attributes, and the creation/destruction events for the Carrier. Note that the names and types of the attributes of the ObjType conform to the name and typing conventions described in Sections 10.4.11 and 10.5.7. ParameterTypeDefinitions have been omitted for clarity. See Related Information Section R5-1 for an example of the corresponding ParameterTypeDefinitions.

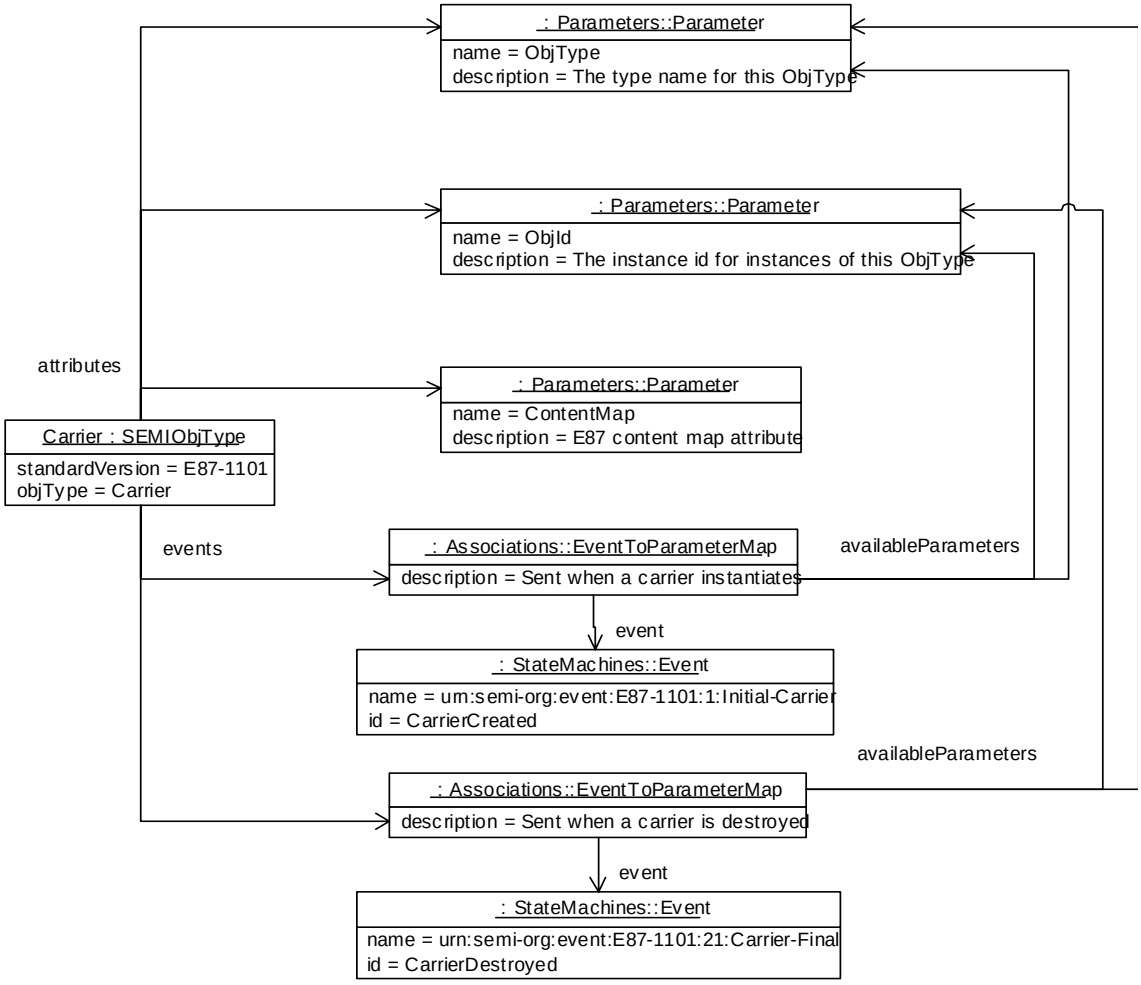

Figure R6-1
E87 Carrier ObjType Example

RELATED INFORMATION 7
STATE MACHINES

NOTICE: This related information is not an official part of SEMI E125 and was derived from the work of the originating committee. This related information was approved for publication by full letter ballot procedures.

R7-1 Examples

R7-1.1 *Simple Pump*

R7-1.1.1 Figure R7-1 shows an example state machine that could be described by the StateMachine mechanism described in Section 10.9. The example is for a very simple vacuum pump that is either pumping down or idle.

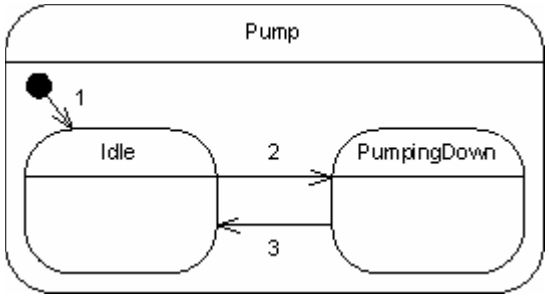

Figure R7-1
Example — Vacuum Pump State Machine

R7-1.1.2 Figure R7-2 shows how the example state machine could be represented using the StateMachine, State, and Transition classes. The top level state, Pump, is represented as a State that contains two substates corresponding to Idle and PumpingDown. Three transitions have been defined, corresponding to the three transitions shown in Figure R7-2.

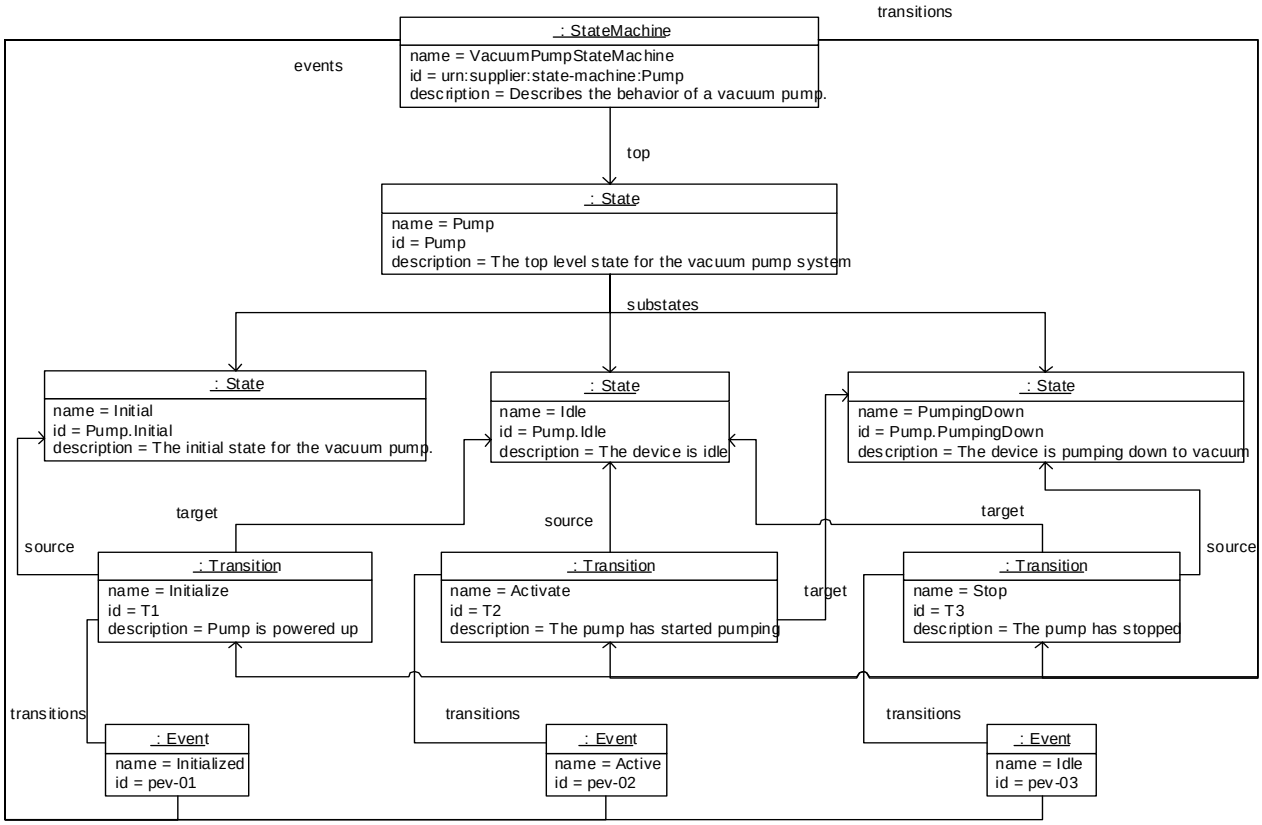

Figure R7-2
Example Representation of the Vacuum Pump State Machine

RELATED INFORMATION 8
R8 UML TERMINOLOGY

NOTICE: This related information is not an official part of SEMI E125 and was derived from the work of the originating committee. This related information was approved for publication by full letter ballot procedures.

R8-1 Glossary

R8-1.1 In order to help with understanding terminology used in conjunction with object technology, this section shares excerpts from the glossary of the OMG standard for UML — the predominant modeling notation.

R8-1.2 These definitions are taken from B.2 Glossary of Terms of version 1.4 of the OMG UML specification, 01-09-67, available from **http://www.omg.org/technology/documents/modeling_spec_catalog.htm**.

R8-1.2.1 *abstract class* — a class that cannot be directly instantiated. Contrast: *concrete class*.

R8-1.2.2 *aggregate [class]* — a class that represents the "whole" in an aggregation (whole-part) relationship. See: *aggregation.*

R8-1.2.3 *aggregation* — a special form of association that specifies a whole-part relationship between the aggregate (whole) and a component part. See: *composition*.

R8-1.2.4 *association* — the semantic relationship between two or more classifiers that specifies connections among their instances.

R8-1.2.5 *attribute* — a feature within a classifier that describes a range of values that instances of the classifier may hold.

R8-1.2.6 *cardinality* — the number of elements in a set. Contrast: *multiplicity*.

R8-1.2.7 *child* — in a generalization relationship, the specialization of another element, the parent. See: *subclass*, *subtype*. Contrast: *parent*.

R8-1.2.8 *class* — a description of a set of objects that share the same attributes, operations, methods, relationships, and semantics. A class may use a set of interfaces to specify collections of operations it provides to its environment. See: *interface*.

R8-1.2.9 *classifier* — a mechanism that describes behavioral and structural features. Classifiers include interfaces, classes, datatypes, and components.

R8-1.2.10 *classification* — the assignment of an object to a classifier. See: *dynamic classification*, *multiple classification, static classification*.

R8-1.2.11 *class diagram* — a diagram that shows a collection of declarative (static) model elements, such as classes, types, and their contents and relationships.

R8-1.2.12 *composition* — a form of aggregation which requires that a part instance be included in at most one composite at a time, and that the composite object is responsible for the creation and destruction of the parts. Composition may be recursive. Synonym: *composite aggregation*.

R8-1.2.13 *concrete class* — a class that can be directly instantiated. Contrast: *abstract class.*

R8-1.2.14 *constraint* — a semantic condition or restriction. Certain constraints are predefined in the UML, others may be user defined. Constraints are one of three extensibility mechanisms in UML. See: *tagged value, stereotype.*

R8-1.2.15 *dependency* — a relationship between two modeling elements, in which a change to one modeling element (the independent element) will affect the other modeling element (the dependent element).

R8-1.2.16 *diagram* — a graphical presentation of a collection of model elements, most often rendered as a connected graph of arcs (relationships) and vertices (other model elements). UML supports the following diagrams: class diagram, object diagram, use case diagram, sequence diagram, collaboration diagram, state diagram, activity diagram, component diagram, and deployment diagram.

R8-1.2.17 *element* — an atomic constituent of a model.

R8-1.2.18 *feature* — a property, like operation or attribute, which is encapsulated within a classifier, such as an interface, a class, or a datatype.

R8-1.2.19 *generalization* — a taxonomic relationship between a more general element and a more specific element. The more specific element is fully consistent with the more general element and contains additional information. An instance of the more specific element may be used where the more general element is allowed. See: *inheritance.*

R8-1.2.20 *inheritance* — the mechanism by which more specific elements incorporate structure and behavior of more general elements related by behavior. See: *generalization*.

R8-1.2.21 *instance* — an entity that has unique identity, a set of operations that can be applied to it, and state that stores the effects of the operations. See: *object*.

R8-1.2.22 *interface* — a named set of operations that characterize the behavior of an element.

R8-1.2.23 *interface inheritance* — the inheritance of the interface of a more general element. Does not include inheritance of the implementation. Contrast: *implementation inheritance*.

R8-1.2.24 *message* — a specification of the conveyance of information from one instance to another, with the expectation that activity will ensue. A message may specify the raising of a signal or the call of an operation.

R8-1.2.25 *method* — the implementation of an operation. It specifies the algorithm or procedure associated with an operation.

R8-1.2.26 *model* [MOF] — an abstraction of a physical system with a certain purpose. See: *physical system*. Usage note: In the context of the MOF specification, which describes a meta-metamodel, for brevity the metametamodel is frequently referred to as simply the model.

R8-1.2.27 *multiple inheritance* — a semantic variation of generalization in which a type may have more than one supertype. Contrast: *single inheritance*.

R8-1.2.28 *multiplicity* — a specification of the range of allowable cardinalities that a set may assume. Multiplicity specifications may be given for roles within associations, parts within composites, repetitions, and other purposes. Essentially a multiplicity is a (possibly infinite) subset of the non-negative integers. Contrast: *cardinality*.

R8-1.2.29 *object* — an entity with a well-defined boundary and identity that encapsulates state and behavior. State is represented by attributes and relationships, behavior is represented by operations, methods, and state machines. An object is an instance of a class. See: *class, instance*.

R8-1.2.30 *object diagram* — a diagram that encompasses objects and their relationships at a point in time. An object diagram may be considered a special case of a class diagram or a collaboration diagram. See: *class diagram, collaboration diagram.*

R8-1.2.31 *operation* — a service that can be requested from an object to effect behavior. An operation has a signature, which may restrict the actual parameters that are possible.

R8-1.2.32 *parent* — in a generalization relationship, the generalization of another element, the child. See: *subclass*, *subtype*. Contrast: *child*.

R8-1.2.33 *package* — a general purpose mechanism for organizing elements into groups. Packages may be nested within other packages.

R8-1.2.34 *qualifier* — an association attribute or tuple of attributes whose values partition the set of objects related to an object across an association.

R8-1.2.35 *relationship* — a semantic connection among model elements. Examples of relationships include associations and generalizations.

R8-1.2.36 *role* — the named specific behavior of an entity participating in a particular context. A role may be static (for example, an association end) or dynamic (for example, a collaboration role).

R8-1.2.37 *single inheritance* — a semantic variation of generalization in which a type may have only one supertype. Synonym: *multiple inheritance* [OMA]. Contrast: *multiple inheritance*.

R8-1.2.38 *subclass* — in a generalization relationship, the specialization of another class; the superclass. See: *generalization*. Contrast: *superclass*.

R8-1.2.39 *subtype* — in a generalization relationship, the specialization of another type; the supertype. See: *generalization*. Contrast: *supertype*.

R8-1.2.40 *superclass* — in a generalization relationship, the generalization of another class; the subclass. See: *generalization.* Contrast: *subclass*.

R8-1.2.41 *supertype* — in a generalization relationship, the generalization of another type; the subtype. See: *generalization*. Contrast: *subtype*.

R8-1.2.42 *type* — a stereotyped class that specifies a domain of objects together with the operations applicable to the objects, without defining the physical implementation of those objects. A type may not contain any methods, maintain its own thread of control, or be nested. However, it may have attributes and associations. Although an object may have at most one implementation class, it may conform to multiple different types. See: *implementation class.* Contrast: *interface*.

R8-1.2.43 *visibility* — an enumeration whose value (public, protected, or private) denotes how the model element to which it refers may be seen outside its enclosing namespace.

SEMI E125.1-0305
PROVISIONAL SPECIFICATION FOR SOAP BINDING FOR EQUIPMENT SELF DESCRIPTION (EqSD)

This provisional specification 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 on December 10, 2005. Initially available at www.semi.org February 2005; to be published March 2005.

Table of Contents

List of Figures

List of Tables

1 Purpose

1.1 *SOAP Implementation Mapping*

1.1.1 The purpose of this specification is to provide an implementation mapping of the SEMI E125 specification to the SOAP 1.1 protocol. This document provides a description of the XML Schema data types used to support the UML classes defined in SEMI E125, and also describes the WSDL port type and binding definitions used to support the operations defined by the interfaces specified in SEMI E125.

2 Scope

2.1 *Specification Scope*

2.1.1 The scope of this specification is the representation of the EqSD model in an XML Schema and corresponding WSDL port types and bindings. It will not add new domain information or concepts to the SEMI E125 model. The only additions made are those needed to render useful WSDL or XML Schema.

2.1.2 This specification requires SEMI E132 authentication, and defines requirements for the application of SEMI E132 concepts to the SEMI E125 specification.

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 *Provisional Specification*

3.1.1 This specification is provisional pending the approval of the SEMI E138, SEMI E120.1 and SEMI E132.1. This specification relies on XML Schema types defined by each of these specifications, and cannot be fully implemented without these types. Finalization of these specifications is a condition for the removal of the provisional status of this document.

3.1.2 This specification is provisional pending approval of the SEMI specification for Units for the Semiconductor Industry. This specification relies on the unit of measure symbols defined in that specification. Finalization of this specification is a condition for the removal of the provisional status of this document.

4 Referenced Standards

4.1 *SEMI Standards*

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

SEMI E120.1 — XML Schema for the Common Equipment Model

SEMI E132.1 — Specification for SOAP Binding of Equipment Client Authentication and Authorization

SEMI E125 — Specification for Equipment Self Description (EqSD)

4.2 *OMG Standards*[1]

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

4.3 *W3C Standards*[2]

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

---

1 Object Management Group, Inc., 250 First Ave. Suite 100, Needham, MA 02494, USA, Telephone:1.781.444.0404, Fax: 1.781.444.0320, website: www.omg.org.

2 World Wide Consortium, Massachusetts Institute of Technology (MIT) Computer Science and Artificial Intelligence Laboratory (CSAIL), 32 Vassar Street Room 32-G515, Cambridge, MA 02139, USA Telephone: 1.617.253.2613 Fax: 1.617.258.5999, website: www.w3c.org.