| *Variable Name* | *Description* | *Type* | *Access* | *Comment* |
|---|---|---|---|---|
| Reject Reason | The reason a reticle is rejected. | Enumerated:<br>ID Verification failedParticle qualification failed<br>Rejected By Host<br>Rejected By Operator<br>Inspection Results Expired | RO | Information to aid the host in understanding why a reticle is rejected. |

[#1] These are defined in the Alarm Information table.

[#2] For all the variables defined as lists of port information (such as states), the port information should be placed in the list in numerical order by PortID, from lowest to highest.

## 22 Alarms

22.1 This section includes specific alarms that are required to be implemented by RPMS compliant equipment.

22.2 *Alarm List Table*

22.2.1 Table 57 is a listing of required alarms for both fixed buffer and internal buffer equipment. This list is only a subset of the pod transfer alarms. There may be more pod transfer related alarms that are not listed here.

**Table 57 Alarm List**

| *Equipment* | | *Danger* | | *Affected* | | |
|---|---|---|---|---|---|---|
| *Configuration* | *Alarm Text* | *Potential* | *Imminent* | *Operator* | *Equipment* | *Material* |
| Fixed & Internal Pod Buffer Equipment | PIO Failure | X | | X | X | X |
| | Access Mode Violation | X | | X | X | X |
| | Attempt To Use Out Of Service Load Port | X | | | X | X |
| | Pod Placement Error | X | | X | X | X |
| | Pod Open/Close Failure | X | | | X | X |
| Fixed and Internal Pod Buffer Equipment | Duplicate PodID | X | | | | X |
| Fixed & Internal Pod Buffer Equipment | Pod Removal Error | X | | X | X | X |
| Duplicate Reticle ID | A duplicate reticle is detected. | X | | | | X |
| Pod Pick/Place error | The internal robot that moves the Pods failed to pick or place a pod. Can be triggered if the state of the system after a pick or place is inconsistent (i.e., both robot and pod think they are empty). | X | | | X | X |
| Reticle Pick/Place error | The internal robot that moves the reticles failed to pick or place a reticle. Can be triggered if the state of the system after a pick or place is inconsistent (i.e. both robot and pod think they are empty). | X | | | X | X |
| Automation Error | Automation errors that do not fit other categories have been detected. | X | | X | X | X |

22.3 *Alarm Usage*

22.3.1 Reticle and Pod Management requires standardization on the alarm codes. Many errors may be mapped to a sinlge "generic" alarm code. In this case, the recovery scenario would be the same but the alarm text shall contain a more descriptive explanation of the error. For example, the Pod Open/Close alarm can be caused by many different opener errors.

22.3.2 This alarm applies to both Fixed and Internal Pod Buffer Equipment.

22.3.3 For any load port Alarm condition, process equipment should not stop processing current jobs.

**Table 58 Alarm Information**

| *Alarm Description* | *Alarm Number* | *Auto Clear* | *Set trigger* | *Clear Trigger* |
|---|---|---|---|---|
| PIO Failure | 1 | NO | E84 error. | Manual intervention |
| Access Mode Violation (unexpected load) | 2 | NO | Operator places pod on load port with access mode set to auto. | Operator switches to manual<br>Operator removes the pod<br>Operator switches to auto<br>Operator clears alarm at UI |
| Access Mode Violation (unexpected unload) | | NO | Operator takes a pod off load port with access mode set to auto. | Operator clears alarm at UI |
| Attempt to use out of service load port | 3 | YES | Pod placed on load port with transfer state set to out-of-service. | Pod is removed |
| Pod placement/placement error | 4 | YES | Sensors detect pod is not properly placed at pod opener. | Sensors detect pod is properly placed at pod opener or no pod present |
| Pod Open/Close error | 5 | NO | Opener errors while opening or closing. | Manual intervention |
| Duplicate Pod ID | 6 | Yes | A duplicate pod is detected. | Methods vary by equipment type |
| Pod Removal Error | 7 | NO | Pod removed before the Pod is in the READY TO UNLOAD state. | Operator clears alarm at UI |
| Duplicate Reticle ID | 8 | YES | A duplicate reticle is detected. | Methods vary by equipment type |
| Pod Pick/Place error | 9 | NO | The internal robot that moves the Pods failed to pick or place a pod. Can be triggered if the state of the system after a pick or place is inconsistent (i.e. both robot and pod think they are empty). | Manual intervention |
| Reticle Pick / Place error | 10 | NO | The internal robot that moves the reticles failed to pick or place a reticle. Can be triggered if the state of the system after a pick or place is inconsistent (i.e. both robot and pod think they are empty). | Manual intervention |
| Automation Error | 11 | NO | Equipment and Alarm dependent, alarms that do not fit other categories. | Manual intervention |

22.4 *Duplicate PodID at Production Equipment*

22.4.1 If the equipment receives a pod with a PodID that is the same as that of another pod present at the equipment, the following rules shall apply:

1. The second pod with a PodID shall not be processed.

2. If processing on the first pod with the PodID has not begun, it should not be processed.

3. If processing on the first pod has begun a Duplicate PodID In Process event shall be issued to notify the host.

## 23 Reticle and Pod Management Equipment Constants

23.1 *Reticle ID Verification Constant* — Reticle ID verification may or may not be required by IC makers. To facilitate common implementation across reticle handling tools, an equipment constant (ECV) called ReticleIDVerification (RIV) is required. The values of this constant are Boolean. If true, ID verification is required. Verification is provided by the equipment if the reticle is instantiated (by service) prior to reticle ID read. Verification is provided by the Host or an operator if the reticle not instantiated at the time of reticle ID read. If false, ID verification is not required.

23.2 Particle inspection may or may not be required by IC makers. To facilitate common implementation across reticle handling tools, an equipment constant call ReticleParticleInspection (RPI) is required. The values of this constant are Boolean. If true, particle inspection is required. If false, no particle inspection is required.

## 24 Requirements for Compliance

24.1 Table 59 provides a checklist for RPMS compliance.

**Table 59 RPMS Compliance Statement**

| *Fundamental RPMS Requirements* | *RPMS Section* | *Implemented* | *RPMS Compliant* |
|---|---|---|---|
| Load Port Numbering | 9.2 | ☐ Yes ☐ No | ☐ Yes ☐ No |
| Reticle Pod Slot Numbering | 9.3 | ☐ Yes ☐ No | ☐ Yes ☐ No |
| Reticle Pod Load Port Transfer State Model | 9.5 | ☐ Yes ☐ No | ☐ Yes ☐ No |
| Reticle Pod Object Implementation | 10 | ☐ Yes ☐ No | ☐ Yes ☐ No |
| Reticle Object Implemenation | 14 | ☐ Yes ☐ No | ☐ Yes ☐ No |
| Reticle State Model | 14.4 | ☐ Yes ☐ No | ☐ Yes ☐ No |
| Reticle Pod Load Port/Pod Association State Model | 13 | ☐ Yes ☐ No | ☐ Yes ☐ No |
| Reticle Location Object Implementation | 17 | ☐ Yes ☐ No | ☐ Yes ☐ No |
| Reticle Location State Model | 17.2 | ☐ Yes ☐ No | ☐ Yes ☐ No |
| PodID Verification Support | 15.2 | ☐ Yes ☐ No | ☐ Yes ☐ No |
| Slot Map Verification Support | 15.3 | ☐ Yes ☐ No | ☐ Yes ☐ No |
| Reticle Verification Support | 15.4 | ☐ Yes ☐ No | ☐ Yes ☐ No |
| Fundamental Services Implementation | 18.4 | ☐ Yes ☐ No | ☐ Yes ☐ No |
| Additional Events Implementation | 20 | ☐ Yes ☐ No | ☐ Yes ☐ No |
| Variable Data Definitions | 21.2 | ☐ Yes ☐ No | ☐ Yes ☐ No |
| Alarms Implementation | 22 | ☐ Yes ☐ No | ☐ Yes ☐ No |
| *Additional RPMS Capabilities* | *RPMS Section* | *Implemented* | *RPMS Compliant* |
| Reservation Visible Signal | 12.2 | ☐ Yes ☐ No | ☐ Yes ☐ No |
| Reticle Pod Load Port Reservation State Model | 12 | ☐ Yes ☐ No | ☐ Yes ☐ No |
| ReticleTransferJob | 18.5.11 | ☐ Yes ☐ No | ☐ Yes ☐ No |
| Access Mode State Model | 11 | ☐ Yes ☐ No | ☐ Yes ☐ No |
| Additional Services Implementation | 18.5 | ☐ Yes ☐ No | ☐ Yes ☐ No |

# APPENDIX 1
# APPLICATION NOTES

**NOTICE**: The material in this appendix is an official part of SEMI E109 and was approved by full letter ballot procedures.

A1-1 This is a place holder for the description of when the Alarms can be cleared.

# SEMI E109.1-0704
# PROVISIONAL SPECIFICATION FOR SECS-II PROTOCOL FOR RETICLE AND POD MANAGEMENT (RPMS)

This 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 Facilities Committee on March 14, 2004. Initially available at www.semi.org May 2004; to be published July 2004. Originally published March 2002; previously published November 2002.

## 1 Purpose

1.1 This document maps the services of SEMI E109 to SECS-II streams and functions. This document also maps the data of SEMI E109 to E5 data definitions.

## 2 Scope

2.1 This is a provisional standard that covers host and equipment communication for equipment that handle reticles. This includes equipment that handle reticles both inside and outside of a reticle pod. The provisional status is required because of the immaturity of implementations of integrated equipment with Automated Material Handling Systems (AMHS) or Automated Reticle Handling Systems (ARHS). Additional specifications may be defined to add further functionality. Also, further exception handling and error recovery scenarios need to be defined.

2.2 This document applies to all implementations of SEMI E109 that use SECS-II message protocol (SEMI E5). Compliance to this standard requires compliance to both SEMI E109 and SEMI E5.

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

## 3 Limitations

3.1 This standard applies to semiconductor equipment that handles reticles and reticle carriers. This standard is intended to be used for lithography production, bare reticle storage, and reticle inspection equipment. It may or may not be applied to other types of equipment.

## 4 Referenced Standards

4.1 *SEMI Standards*

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

SEMI E39.1 — SECS-II Protocol for Object Services Standard (OSS)

SEMI E109 — Provisional Specification for Reticle and Pod Management (RPMS)

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

## 5 Services Mapping

5.1 This sections shows the specific SECS-II streams and functions that shall be used for SECS-II implementations of the services defined in SEMI E109, as well as the parameter mapping for data attached to services.

5.2 *Services Message Mapping*

5.2.1 Table 1 defines the relationships between SEMI E109 services and SECS-II messages.

**Table 1 Service Message Mapping**

| *Service Name* | *Stream/Function* | *SECS-II Message Name* |
|---|---|---|
| Bind | S3F17/F18 | Carrier Action Request/Acknowledge |
| CancelAllPodOut | S3F33/F34 | Cancel All Pod Out Request/Acknowledge |
| CancelBind | S3F17/F18 | Carrier Action Request/Acknowledge |
| CancelMoveReticle | S14F19/20 | Generic Service Request/Acknowledge |
| CancelPod | S3F17/F18 | Carrier Action Request/Acknowledge |
| CancelPodAtPort | S3F17/F18 | Carrier Action Request/Acknowledge |
| CancelPodNotification | S3F17/F18 | Carrier Action Request/Acknowledge |
| CancelPodOut | S3F17/F18 | Carrier Action Request/Acknowledge |
| CancelReservationAtPort | S3F25/F26 | Port Action Request/Acknowledge |
| CancelReticleTransferJob | S3F35/F36 | Carrier Action Request/Acknowledge |
| ChangeAccess | S3F27/F28 | Change Access/Acknowledge |
| ChangeServiceStatus | S3F25/F26 | Port Action Request/Acknowledge |
| Clamp | S3F17/F18 | Carrier Action Request/Acknowledge |
| ClosePod | S3F17/F18 | Carrier Action Request/Acknowledge |
| IndexDown | S3F17/F18 | Carrier Action Request/Acknowledge |
| IndexUp | S3F17/F18 | Carrier Action Request/Acknowledge |
| MoveReticle | S14F19/F20 | Generic Service Request/Acknowledge |
| OktoUseReticle | S14F19/F20 | Generic Service Request/Acknowledge |
| PodComplete | S3F17/F18 | Carrier Action Request/Acknowledge |
| PodIn | S3F17/F18 | Carrier Action Request/Acknowledge |
| PodNotification | S3F17/F18 | Carrier Action Request/Acknowledge |
| PodOut | S3F17/F18 | Carrier Action Request/Acknowledge |
| PodRelease | S3F17/F18 | Carrier Action Request/Acknowledge |
| PodTagReadData | S3F29/F30 | Carrier Tag Read Data Request/Acknowledge |
| PodTagWriteData | S3F31/F32 | Carrier Tag Write Data Request/Acknowledge |
| ProceedWithPod | S3F17/F18 | Carrier Action Request/Acknowledge |
| ProceedWithReticle | S14F19/20 | Generic Service Request/Acknowledge |
| RejectReticle | S14F19/F20 | Generic Service Request/Acknowledge |
| Re-qualifyReticle | S14F19/F20 | Generic Service Request/Acknowledge |
| ReserveAtPort | S3F25/F26 | Port Action Request/Acknowledge |
| ReticleTransferJob | S3F35/F36 | Reticle Transfer Job Request/Acknowledge |
| SetQualificationIntervalTime | S14F3/4 | SetAttr Request |
| Unclamp | S3F17/F18 | Carrier Action Request/Acknowledge |

## 5.3 *Services Parameter Mapping*

5.3.1 Table 2 maps the SEMI E109 service parameters to SECS-II Data Items.

NOTE 1: Use of parameters not specified for a given message in SEMI E109 is prohibited. SECS-II data items not used for a given message shall be sent as zero length items.

**Table 2 Service Parameters to SECS-II Data Item Mapping**

| *Parameter Name* | *Parameter Range* | *SECS-II Data Item* |
|---|---|---|
| AccessMode | Enumerated:<br>0 = MANUAL<br>1 = AUTO | ACCESSMODE |

| *Parameter Name* | *Parameter Range* | *SECS-II Data Item* |
|---|---|---|
| ArrivingReticleList | List of n structures, where<br>n = capacity<br>Reticle ID<br>ReticleRemovalinstruction<br>ReticlePropertiesList<br><br>When a slot in a pod contains no reticle, the value for Reticle ID should be null, the ReticleRemovalInstruction$_n$ should be PASS BY, and the ReticlePropertiesList, should be a zero length list. | L,n<br>1. L,3<br>1.<RETICLEID$_1$><br>2.< RETREMOVEINSTR$_1$><br>3. L,m<br>1. L,2<br>1. <ATTRID$_{1,1}$><br>2. <ATTRDATA$_{1,1}$><br>:<br>m. L,2<br>1.<ATTRID$_{1,m}$><br>2.<ATTRDATA$_{1,m}$><br>:<br>n. L,3<br>1.<RETICLEID$_n$><br>2. <RETREMOVEINSTR$_n$><br>3. L,m<br>1. L,2<br>1. <ATTRID$_{n,1}$><br>2. <ATTRDATA$_{n,1}$><br>:<br>m. L,2<br>1. <ATTRID$_{n,m}$><br>2. <ATTRDATA$_{n,m}$> |
| CMAcknowledge | Enumerated | CAACK |
| CMStatus | Structure | L,2<br>1.<CAACK><br>2.Status |
| Data | 1–80 characters | DATA |
| DataSeg | 1–80 characters | DATASEG |
| DataSize | 0–65,535 (integer) | DATALENGTH |
| DepartingReticleList | List of n structures, where<br>n = capacity<br>Reticle ID<br>ReticlePlacementInstruction<br>When a slot in a pod contains no reticle and will not have a reticle placed, the value for Reticle ID should be null and the ReticlePlacmentInstruction$_n$ should be PASS BY. | L,n<br>1. L,2<br>1.<RETICLEID$_1$><br>2.<RETPLACEINSTR$_1$><br>:<br>n. L,2<br>1.<RETICLEID$_n$><br>2.<RETPLACEINSTR$_n$> |
| DestinationLocation | 1–80 characters | RPMDESTLOC |
| ErrorCode | Enumerated | ERRCODE |
| ErrorText | 1–80 characters | ERRTEXT |
| InputPortID | U1(0, 101–255) | INPTN |
| OutputPortID | U1(0, 101–255) | OUTPTN |
| PodAttributeData | Any | ATTRDATA |
| PodAttributeID | 1–40 characters | ATTRID |
| PodID | 1–64 characters | PODID |

| *Parameter Name* | *Parameter Range* | *SECS-II Data Item* |
|---|---|---|
| PodPropertiesList | List of AttributeID and AttributeData | L,n<br>1.L,2<br>1.<ATTRID$_1$><br>2.<ATTRDATA$_1$><br>:<br>n.L,2<br>1.<ATTRID$_n$><br>2.<ATTRDATA$_n$> |
| PortID | U1(101–255) | PTN |
| PortList | List of PortID | L,n<br>1.<PTN$_1$><br>:<br>n.<PTN$_n$> |
| ReticleAttributeData | Any | ATTRDATA |
| ReticleAttributeID | 1–40 characters | ATTRID |
| ReticleLocation-AttributeData | Any | ATTRDATA |
| ReticleLocation-AttributeID | 1–40 characters | ATTRID |
| ReticlePlacement-Instruction | Enumerated:<br><br>0 = PLACE<br>1 = PASS BY<br>2 = CURRENTLY OCCUPIED | RETPLACEINSTR |
| ReticlePodLocationID | 1–80 characters | CATTRID |
| ReticleRemoval-Instruction | Enumerated:<br>0 = REMOVE<br>1 = PASS BY | RETREMOVEINSTR |
| ReticleID | 1–64 characters | RETICLEID |
| ReticleLocation-PropertiesList | List of AttributeID and AttributeData | L,n<br>1.L,2<br>1.<ATTRID$_1$><br>2.<ATTRDATA$_1$><br>:<br>n.L,2<br>1.<ATTRID$_n$><br>2.<ATTRDATA$_n$> |
| ReticlePropertiesList | List of AttributeID and AttributeData | L,n<br>1.L,2<br>1.<ATTRID$_1$><br>2.<ATTRDATA$_1$><br>:<br>n.L,2<br>1.< ATTRID$_n$><br>2.< ATTRDATA$_n$> |

| *Parameter Name* | *Parameter Range* | *SECS-II Data Item* |
|---|---|---|
| RPMAcknowledge | Enumerated:<br>0 = Acknowledge, service has been performed<br>1 = Service does not exist<br>2 = Cannot perform now<br>3 = At least parameter is invalid<br>4 = Acknowledge, service will be performed with completion notified later with parameters for response<br>5 = Service is not completed or prohibited<br>6 = no such objects exist<br>7–63 Reserved | SVCACK |
| RPMStatus | List of RPMAcknowledge and Status | L,2<br>1.<SVCACK><br>2. L,n<br>1. L,2<br>1. <ERRCODE$_1$><br>2. <ERRTEXT$_1$><br>:<br>n. L,2<br>1. <ERRCODE$_n$><br>2. <ERRTEXT$_n$> |
| ServiceStatus | "ServiceStatus" | PARAMNAME |
| ServiceStatusValue | Enumerated:<br>OUT OF SERVICE = 0<br>IN SERVICE = 1 | PARAMVAL = U1 |
| SourceLocation | 1–80 characters | RPMSOURLOC |
| Status | List of n errors | L,n<br>1.L,2<br>1.<ERRCODE$_1$><br>2.<ERRTEXT$_1$><br>:<br>n.L,2<br>1.<ERRCODE$_n$><br>2.<ERRTEXT$_n$> |

5.4 *Object Service Parameter Mapping*

5.4.1 Table 3 maps the SEMI E109 service parameters that use S14F19 to SECS-II data items.

**Table 3 Object Service Parameters for SECS-II**

| *Parameter Name* | *Format* | *SECS-II Data Item* | *SECS-II Format* |
|---|---|---|---|
| ObjectAction | "CancelMoveReticle"<br>"MoveReticle"<br>"OktoUseReticle"<br>"RejectReticle"<br>"Re-qualifyReticle" | SVCNAME | 20 |
| ObjectAction-ParameterRequest | List of one or more arguments (parameters) providing additional information concerning the action requested. Structure composed of ServiceParameterName and ServiceParameterValue | L,n<br>1. L,2<br>1. <SPNAME$_1$><br>2. <SPVAL$_1$><br>:<br>m. L,2<br>1. <SPNAME$_m$><br>2. <SPVAL$_m$> | |

| *Parameter Name* | *Format* | *SECS-II Data Item* | *SECS-II Format* |
|---|---|---|---|
| ObjectActionParameter Result | List on one or more arguments (parameters) providing additional information concerning the result of the action requested. | L,n<br>1. L,2<br>1. <$SPNAME_1$><br>2. <$SPVAL_1$><br>:<br>m. L,2<br>1. <$SPNAME_m$><br>2. <$SPIVAL_m$> | |
| RPMAcknowledge | Enumerated | SVCACK | 10 |
| ServiceParameterName | " ReticleID"<br>" PortID"<br>" DestinationLocation"<br>" SourceLocation"<br>" QualificationIntervalTime"<br>“Status” | SPNAME | 20 |
| ServiceParameterValue | Depends on parameter used | SPVAL | 10, 20, 51, 52 |
| Status | List of one or more arguments (parameters) providing information concerning the result of the action requested. Structure composed of ErrorText and ErrorCode. | L,m<br>1. L,2<br>1. <$ERRCODE_1$><br>2. <$ERRTEXT_1$><br>:<br>m. L,2<br>1. <$ERRCODE_m$><br>2. <$ERRTEXT_m$> | |
| ErrorCode | Enumerated | ERRCODE | 51 |
| ErrorText | Text | ERRTEXT | 20 |

5.5 *SECS-II Data Items without Corresponding E109 Parameters*

5.5.1 Table 4 contains the SECS-II data items that do not correspond to SEMI E109 service parameters.

**Table 4 Additional Data Item Requirements Table**

| *Function* | *SECS-II Data Item* |
|---|---|
| Used by S3,F17 to differentiate between “Bind”, “CancelPod”, “CancelPodNotification”, “CancelPodOut”, “CancelPodAtPort”, “CancelBind”, “Clamp”, “ClosePod”, “IndexDown”, “IndexUp”, “OpenPod”, “PodComplete”, “PodIn”, “PodNotification”, “PodOut”, “PodRelease”, “PodTagReadData”, “PodTagWriteData”, “ProceedWithPod”, and “Unclamp” services.<br>Note: the text between the quotes is the actual value used in the parameter CARRIERACTION. | CARRIERACTION |
| Used to satisfy SECS-II conventions for linking a multi-block inquiry with a subsequent multi-block message. Neither required nor specified by RPMS. | DATAID |
| Used to inform receiver of total message length size for SECS-II multi-block conventions. Neither required nor specified by SEMI E109. | DATALENGTH |
| Used to satisfy SECS-II multi-block requirements. Neither required nor specified by SEMI E109. | GRANT |

| *Function* | *SECS-II Data Item* |
|---|---|
| Used by S3F25 to differentiate between port related “ChangeServiceStatus,” “CancelReservationatPort,” and “ReserveAtPort” services. Note: the text between the quotes is the actual value used in the parameter PORTACTION. | PORTACTION |
| Used by S3F27 to specify desired Access mode. | ACCESSMODE |
| Used by S3F35 to differentiate between “ReticleTransferJob” and “CancelReticleTransferJob” | JOBACTION |

## 6 Variable Data Item Mapping

6.1 This sections shows the specific SECS-II data classes, and formats needed for the SECS-II implementation of SEMI E109 data items.

**Table 5 Variable Data Item Mapping Table**

| *Variable Name* | *Class* | *Format* |
|---|---|---|
| Access Mode | DVVAL | 51 (U1) |
| Access Mode$_i$ | SV | 51 (U1) |
| AlarmInfo | DVVAL | L,2<br>1. Alarm number (U4)<br>2. Alarm Description (A) |
| AvailPartitionCapacity | DVVAL | 51 (U1) |
| AvailPartitionCapacity$_i$ | SV | 51 (U1) |
| BufferCapacityList | SV | L,n<br>1.<BufferPartitionInfo$_i$><br>.<br>.<br>n.<BufferPartitionInfo$_i$> |
| BufferPartitionInfo | DVVAL | L,4<br>1.<PartitionID><br>2.<PartitionType><br>3.<AvailPartionCapacity><br>4.<PartitionCapacity> |
| BufferPartitionInfo$_i$ | SV | L,4<br>1.<PartitionID$_i$><br>2.<PartitionType$_i$><br>3.<AvailPartionCapacity$_i$><br>4.<PartitionCapacity$_i$> |
| BypassReadID | DVVAL | 11 (BOOLEAN) |
| PodID | DVVAL | 20 (A[1–64]) |
| PodID$_i$ | SV | 20 (A[1–64]) |
| PodLocationMatrix | SV | L,n<br>1.L,2<br>1.<LocationID$_1$><br>2.<PodID$_1$><br>.<br>.<br>n.L,2<br>1.<LocationID$_n$><br>2.<PodID$_n$> |

| *Variable Name* | *Class* | *Format* |
|---|---|---|
| LocationID | DVVAL | 20 (A[1–64]) |
| $LocationID_i$ | SV | 20 (A[1–64]) |
| PartitionCapacity | DVVAL | 51 (U1) |
| $PartitionCapacity_i$ | SV | 51 (U1) |
| PartitionID | DVVAL | 20 (A[1–64]) |
| $PartitionID_i$ | SV | 20 (A[1–64]) |
| PartitionType | DVVAL | 20 (A[1–64]) |
| $PartitionType_i$ | SV | 20 (A[1–64]) |
| PortAssociationState | DVVAL | 51 (U1)<br>Enumerated as:<br>0 = NOT ASSOCIATED<br>1 = ASSOCIATED |
| $PortAssociationState_i$ | SV | 51 (U1)<br>Enumerated<br>0 = NOT ASSOCIATED<br>1 = ASSOCIATED |
| PortAssociationStateList | SV | L,n<br>1.<$PortAssociationState_i$><br>.<br>.<br>n <$PortAssociationState_n$> |
| PortID | DVVAL | 51 (U1) |
| $PortID_i$ | SV | 51 (U1) |
| PortStateInfo | DVVAL | L,2<br>1.<PortAssociationState><br>2. <PortTransferState> |
| $PortStateInfo_i$ | SV | L,2<br>1.<$PortAssociationState_i$><br>2. <$PortTransferState_i$> |
| PortStateInfolist | SV | L,n<br>1.<$PortStateInfo_1$><br>.<br>.<br>n.<$PortStateInfo_n$> |
| PortTransferState | DVVAL | 51 (U1) |
| $PortTransferState_i$ | SV | 51 (U1) |
| PortTransferStateList | SV | L,n<br>1.<$PortTransferState_1$><br>.<br>.<br>n.<$PortTransferState_n$> |
| Reason | DVVAL | 51 (U1) |
| ReticleIDVerification | ECV | 11 (BOOLEAN) |
| ReticleParticleInspection | ECV | 11 (BOOLEAN) |

**Table 6 Reticle Pod Object Attribute Definitions**

| *Attribute Name* | *Attribute Data Form: SECS-II Structure* |
|---|---|
| "ObjType | 1. "ReticlePod" |
| "ObjID" | 1. <PODID> (Conforms to the restrictions of ObjID as specified in SEMI E39.1, Section 6.) |
| "Capacity" | 51 (U1) Capacity<br>Capacity Range: 1..6<br>Capacity Examples: 1, 6 |
| "ReticlePodAccessingStatus" | 51 (U1) ReticlePodAccessingStatus<br>ReticlePodAccessingStatus enumerated per Variable ReticlePodAccessingStatus<br>0: "NOT AVAILABLE"<br>1: "AVAILABLE"<br>2: "IN ACCESS"<br>3: "COMPLETE" |
| "PodIDStatus" | 51 (U1) PodIDStatus<br>PodIDStatus enumerated per Variable PodIDStatus<br>0: "ID NOT READ"<br>1: "WAITING FOR HOST ON ID STATUS"<br>2: "ID VERIFICATION OK"<br>3: "ID VERIFICATION FAILED" |
| "ContentMap" | L,n n = Capacity<br>1. ReticleID<br>n. ReticleID |
| "ReticlePodLocationID" | 20 (A) ReticlePodLocationID<br>ReticlePodLocationID conforms to the restrictions of ObjID as specified in SEMI E39.1, Section 6. |
| "ReticlePodLockingStatus" | 51 (U1) ReticlePodLockingStatus<br>ReticlePodLockingStatus enumerated per VariableReticlePodLockingStatus<br>0: "NOT LOCKED"<br>1: "RELEASED AND LOCKED"<br>2: "HOLD" |
| "SlotMap" | L,n n = Capacity<br>1.51 (U1) enumerated<br>…<br>n.51 (U1) enumerated<br>Enumerated<br>0: "UNDEFINED"<br>1: "EMPTY"<br>2: "CORRECTLY OCCUPIED" |
| "SlotMapStatus" | 51 (U1) SlotMapStatus<br>SlotMapStatus enumerated per Variable SlotMapStatus.<br>0: "SLOT MAP NOT READ"<br>1: "WAITING FOR HOST ON SLOT MAP STATUS"<br>2: "SLOT MAP VERIFICATION OK"<br>3: "SLOT MAP VERIFICATION FAILED" |

**Table 7 Reticle Object Attribute Definition**

| *Attribute Name* | *Attribute Data Form: SECS-II Structure* |
|---|---|
| "ObjType" | 1. "Reticle" |
| "ObjID" | 1. <RETICLEID> (Conforms to the restrictions of ObjID as specified in SEMI E39.1, Section 6.) |
| # of Exposures | 51 (U1) # of Exposures |
| QualificationIntervalTime | 52 (U2) 0–99999 / 0 is not limit. |

| *Attribute Name* | *Attribute Data Form: SECS-II Structure* |
|---|---|
| QualificationTerminationTime | YYYYMMDDHHMMSSCC |
| ReticleStatus | 51 (U1) ReticleStatus<br>0 “RETICLE NOT PRESENT”<br>1 “WAITING QUALIFICATION”<br>2 “READING ID”<br>3 “QUALIFIED”<br>4 “IN USE”<br>5 “REJECTED”<br>6 “RETICLE WAITING FOR HOST<br>7 “ PARTICLE INSPECTION” |
| ReticleAllocationStatus | 51 (U1) RetcileAllocationStatus<br>0 “NOT ALLOCATED”<br>1 “ALLOCATED” |
| ReticleLocationID | 20 (A) ReticleLocationID<br>ReticleLocationID conforms to the restrictions of ObjID as specified in SEMI E39.1, Section 6. |
| ReticleType | Equipment dependent |

# SEMI E120-0705
# SPECIFICATION FOR THE COMMON EQUIPMENT MODEL (CEM)

This specification was technically approved by the global Information & Control Committee. This edition was approved for publication by the global Audits and Reviews Subcommittee on April 7, 2005. It was available at www.semi.org June 2005 and on CD-ROM in July 2005. Originally published March 2003; previously November 2004.

## 1 Purpose

1.1 The purpose of this specification is to provide a generally applicable object model of semiconductor equipment structure. This object model is intended to be used in the following ways:

- As a guide to equipment suppliers on how to represent the external view of their equipment to the factory host.
- As a base model to be used and extended by other SEMI equipment communication standards.
- As a reference for the creation of technology-specific object model definitions, such as XML schema.

## 2 Scope

2.1 This specification provides a common set of constructs that can be used by the equipment manufacturer to model their equipment structure.

2.2 This specification defines the classes, attributes, relationships, and other detail appropriate for the model of an equipment's structure as viewed by a factory host application through the available communication interfaces.

2.3 The model contained in this specification documents features of the equipment structure considered necessary for more than one (current or future) equipment communication standards.

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

## 3 Limitations

3.1 This specification does not define "object diagrams" (that is, instances) of equipment. These are the responsibility of the equipment supplier to define in conformance with the model.

3.2 This specification does not define the behavior of objects/classes. Behavior definition of classes defined in this specification is left to other equipment communication standards.

3.3 This specification does not define any specific messaging between equipment and factory host. It is expected that communication related to the Common Equipment Model will be defined by other standards.

3.4 This specification addresses only the host view of the equipment. No other part of the factory is modeled by this document.

## 4 Referenced Standards and Documents

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

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

## 5 Terminology

5.1 *Abbreviations and Acronyms*

5.1.1 *CEM* — Common Equipment Model

5.2 *Definitions*

5.2.1 UML terms used in this document conform to definitions in the Glossary of Terms of the OMG specification of UML (referenced above). Related Information, §R1, contains selected definitions from that Glossary of Terms.

5.2.2 Definitions or descriptions of many of the terms used in this specification can be found in the SEMI Compilation of Terms, available on the SEMI web site, http://www.semi.org/.

5.2.3 When terms from the above two sources conflict, the definition from the OMG UML specification takes precedence in this specification.

## 6 Conventions

6.1 This section defines conventions followed in this document.

6.2 *Object Modeling*

6.2.1 *Unified Modeling Language (UML)* — This specification uses UML notation for all class diagrams and for object diagrams provided as examples. No other types of diagrams are used in this specification.

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

6.2.2 *Behavior Not Defined* — This specification, defines classes, their relationships, and attributes. No behavior is specified either in the form of UML state, activity, sequence diagrams, or in the form of operations definitions.

6.2.3 *Name of a Class* — The text capitalizes class names.

6.2.4 *Abstract and Concrete Classes* — Each class is specified as Abstract or Concrete. Abstract classes are not directly implemented. In this specification "implemented" means represented to the factory through the communications interface. All classes defined as concrete may be directly implemented. This specification does not attempt to define equipment control system implementation. In UML class diagrams, abstract class names are shown in *italics*.

6.2.5 *Class Attribute Definition* — The attributes of a class are defined in table format as illustrated by Table 1 below. Note that the "+" sign in front of attributes in UML class diagrams indicates "public" attributes. All attributes defined in this specification are public.

**Table 1 Attribute Table Format**

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

6.2.5.1 *Access* — Attributes may be settable (ReadWrite or RW) or not settable (ReadOnly or RO) through an interface to the equipment.

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

6.2.5.3 *Form* — Defines the data type of the attribute. Data types specified in the Common Equipment Model specification are described in the subsections that follow. Data types in this specification are high-level definitions and should be mapped to the data types of a specific technology as appropriate. Extensions to the Common Equipment Model are not limited to the list of data types defined in this specification. See the definition of "Form" in the SEMI Compilation of Terms for other Form values commonly used in SEMI documents.

6.2.5.3.1 *Enumeration* — A list of named values used as the range of a particular attribute type. In this document, these values are represented as text strings, but may be implemented differently (e.g. as integers that correspond to the named values).

6.2.5.3.2 *String* — A text string. Limitations on length are left to the implementation technology unless otherwise specified in this document.

6.2.5.3.3 *List of xxx* — An item that can hold multiple instances of a specified type (where xxx is the data type).

6.2.6 *Constraints*

6.2.6.1 Constraints in UML are shown with dotted lines, typically with a note attached to explain the constraint. Constraints shown in this way tend to clutter large UML diagrams. Therefore, in this specification, constraints are shown only on the diagrams focusing on individual classes.

6.2.7 *Association Navigability*

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

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

6.2.7.3 From any object, it must be possible to obtain a reference to the target objects of any of its navigable associations.

6.2.7.4 The method of referencing the target objects is implementation dependent and is not specified in this document.

6.2.7.5 Any implementer or subordinate standards mapping the concepts of this specification to an implementation method shall specify the means of referencing the target object of an association from the source object.

6.3 *Model Extension*

6.3.1 Any concrete class in the CEM model can be extended. Extension is achieved through association. In "extension by association" the new capability is contained in a separate class that is related to the original by an association. In the case of CEM, an "*Extension*" class has been added to the model (see ¶8.5.3.4) along with the needed association.

6.3.2 The *Extension* class is associated with *Nameable*, so that any *Nameable* can have multiple *Extensions*. The *Extension* class has no attributes or services and does not itself add anything to the extended CEM class. To have a meaningful extension, a subclass of *Extension* must be created containing the new features.

6.3.3 So, in order to create an extension to a selected concrete CEM class, a new class would be designed with added features. For instance, it might contain a new attribute. This new class would be designed as a subclass of *Extension*. Remember that any *Nameable* can navigate the association to any of its *Extension*s. Therefore, the new subclass of *Extension* could be associated with any CEM class. See Figure 1 below for an illustration. The final step is to associate the new *Extension* subclass to the specific CEM class it is extending.

6.3.4 Most extensions will be designed to work on specific CEM classes. The CEM model does not enforce matching an extension with its intended CEM class. Therefore, the specification of the *Extension* class must describe its allowed usage.

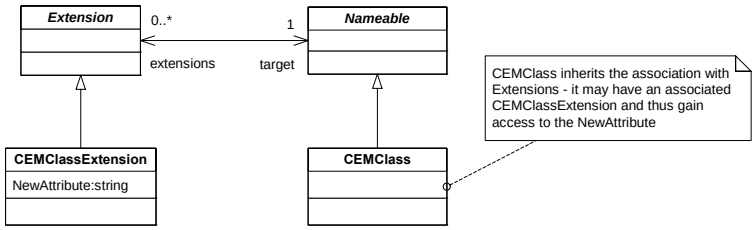



**Figure 1**
**CEM Model Extension**

6.3.5 The scope of CEM is limited to static information about the equipment. However, that limitation does not apply to extensions. So extensions might include static or dynamic attributes as well as links to state models or other complex constructs. An example of an extension that might be added by a supplier is a class containing "manufacturedDate" and "serviceTagNumber" attributes to be applied to an *Equipment* object.

## 7 Background

7.1 This specification documents common characteristics of equipment in a single specification that can be referenced and used by other standards. These other standards are expected to extend the Common Equipment Model to add characteristics unique to that standard. Thus, this specification attempts to be the core of a single, coordinated object model of the equipment with the full model being a composite of multiple standards.

7.2 *Approach to Model Description*

7.2.1 *Class Model vs. Instance Model*

7.2.1.1 A model showing the specifics about particular equipment is considered an "instance model". Each equipment instance model will have its own characteristics based on its function. No single model can directly describe all such instances. Instead, the class model contained in this specification defines the classes to be used to describe the equipment's structure and how they shall relate to one another. The instance model of the equipment is to be provided by the equipment supplier as specified by the CEM class model.

7.2.2 *Model Extensibility*

7.2.2.1 The Common Equipment Model is intended to be extensible (within defined limits). ¶6.3 describes acceptable ways to extend the model.

7.2.2.2 The instance model of the equipment shall thus be constructed by the equipment supplier based on

- the classes defined in this document,
- extensions to this model defined in other SEMI standards, and
- extensions to the standardized model made by the supplier to address unique equipment characteristics.

7.2.3 *External View — Not Equipment Design*

7.2.3.1 There is no expectation that equipment suppliers will use the model specified in this document as the basis for the design of their equipment. The CEM is intended to provide a structure to promote clear communication. It needs to provide an unambiguous view to the factory host that is representative of the equipment and meets the needs of the host.

7.2.4 *Communication Not Defined*

7.2.4.1 This specification leaves to other SEMI standards the definition of how the Common Equipment Model will be communicated (in whole or in part) between the equipment and the factory host. It is to be expected that there will be many uses for the model and the details of communication are best defined in context with planned use.

7.2.5 *Behavior Not Defined*

7.2.5.1 In order to keep the model simple, this document does not define behavior of classes in the CEM. Class behavior is related to function, and thus to functionality supplied by the equipment interface. This functionality and behavior is left to other SEMI standards to define.

7.2.6 *Applicability*

7.2.6.1 Though the CEM attempts to address all semiconductor related factory equipment, it is recognized that this may not be possible. The CEM covers only manufacturing equipment (i.e. equipment that perform their function on material—e.g. no independent environmental monitoring systems, etc.). The vast majority of semiconductor factory equipment should be able to comply with this specification.

7.3 *Uses For The CEM*

7.3.1 In order to understand the CEM, it is necessary to understand expected uses of the model. While it is not possible to anticipate every possible use, the following list presents several expected uses that may be defined by other standards.

7.3.2 *Equipment Self-Description Using Metadata* — An important consideration for the integration of semiconductor equipment into the factory is the ease with which the equipment's structure and capabilities can be understood and translated into functioning software. With traditional semiconductor communication technologies this is achieved by first reading the equipment supplier's communication manual and then configuring or coding

factory software to match the documentation. Unfortunately, manuals are too often incomplete or out of date. If the equipment can provide its instance model (and the extended class model on which it is based) to the host, it is more likely to be complete and accurate. If it can construct that model in real-time, based on its current hardware configuration it will be more accurate. This self-descriptive data is also known as "metadata".

7.3.3 *Relation Of Data To The Equipment* — Data gathered from equipment gains greater meaning if it is clear as to what part of the equipment it pertains and what related data items provide context information. The instance model can be a reference in the definition of the data collection plans and reflected by actual data collected.

7.3.4 *Relation Of Parameters & Setting To The Equipment* — Proper and predictable control of equipment is possible only if the effects of all available parameters and settings is understood. These must be related to the equipment and to the measurable values that they affect. This information can be provided by the instance model.

7.3.5 *Understanding Equipment Capabilities* — The process capabilities of equipment depend on the modules present and the characteristics and status of those modules. In some situations, the hardware configuration of equipment can change on a regular basis.

## 8 Common Equipment Model Definition

8.1 This section documents the Common Equipment Model.

8.2 *Overview*

8.2.1 Documentation of the Common Equipment Model begins with system views of the model and moves to detailed descriptions of the classes with their attributes and relationships. No behavior is defined—that is, no state models and no methods are described for classes.

8.2.2 It is important to note that the two system views presented are of the same model. They are separated to avoid large, complex diagrams, thus enhancing understandability.

8.2.3 The model shows only classes and their relationships. Actual equipment will have corresponding instance models that define the detail of the equipment as viewed through an equipment interface. Thus, the Common Equipment Model in this document provides rules for constructing the instance model of an equipment.

8.2.4 The Common Equipment Model is represented in two system level views. The first is the inheritance hierarchy that shows which classes inherit characteristics (e.g. attributes and relationships) from which other classes. The second view represents the relationships among classes to show general equipment structure. These views are described in general terms to highlight important aspects of the model.

8.2.5 Everything in these two views is repeated in subsequent sections dealing with individual classes in the model. In the individual class views, class of interest is shown with all relationships to other classes. No other detail is shown in these views.

8.3 *Inheritance Hierarchy*

8.3.1 The inheritance hierarchy view, Figure 2, shows the superclass/subclass relationships of the Common Equipment Model.

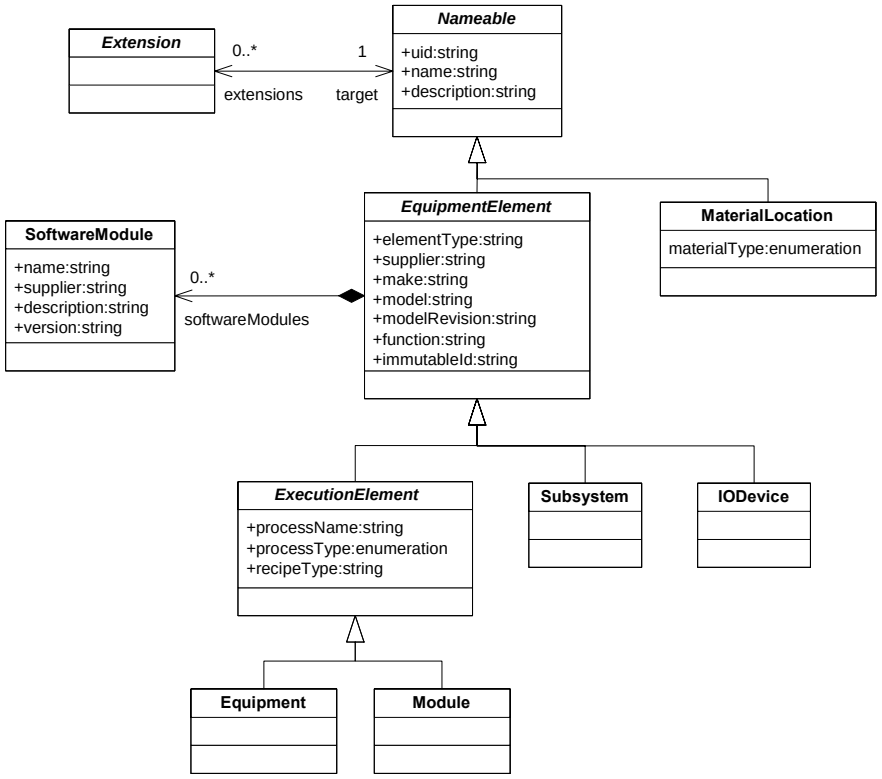



**Figure 2**
**Inheritance Hierarchy**

8.3.2 Items to note about the inheritance hierarchy view:

- Nameable is at the top of the inheritance hierarchy and provides common accessibility to all classes in the Common Equipment Model.
- *Equipment* and *Modules* may be able to process material, but *Subsystems* cannot.
- Note that a subclass inherits the attributes of its superclass. Some subclasses have no additional attributes defined in this specification. It is anticipated that other standards will extend this model to add attributes, relationships, state models, etc. as equipment behavior is mapped onto this model.

8.4 *Relationship View*

8.4.1 The relationship view is shown in Figure 3. This view illustrates expected equipment. Items to note about the relationship view:

- Flexibility is built into the structure so that, for instance, *IODevices* may belong directly to almost any level of the equipment structure.
- *Subsystems*, *Modules*, and *Equipment* may have MaterialLocations (i.e. hold material).
- *Equipment — Equipment* aggregation allows Equipment that acts on behalf of other *Equipment* to describe these *Equipment.*

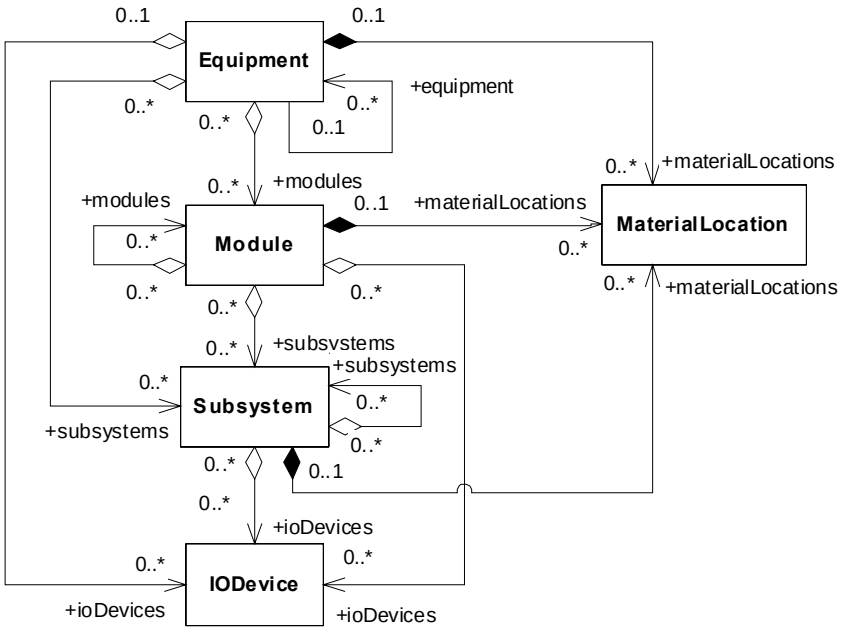



**Figure 3**
**Relationship View**

8.5 *Class Definitions*

8.5.1 Class definitions consist of a brief description of the highlighted class, a class diagram showing the class and all related classes (both from an inheritance and relationship standpoint), a table defining the attributes, and further text as necessary to fully define the class.

8.5.2 For clarity, the abstract classes are listed separately from concrete classes.

8.5.3 *Abstract Classes*

8.5.3.1 *Nameable* — The root of each class in the CEM class hierarchy. This class provides a means for uniquely identifying each component of the equipment structure. *Nameable* is a superclass of *EquipmentElement* and *MaterialLocation* (see Figure 4). It includes three attributes: *uid, name, and description* (see Table 2). *Nameable* is also associated with the class *Extension*. See ¶6.3 for more information on the use of *Extension.*

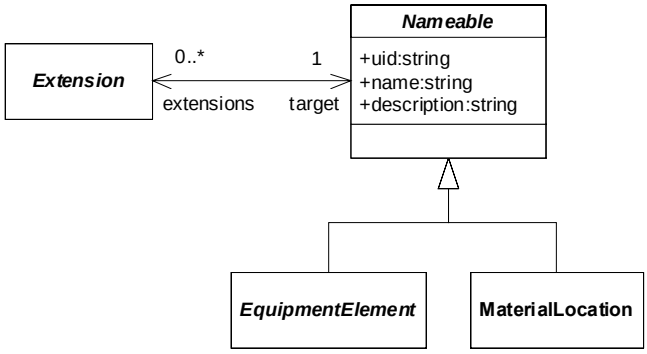



**Figure 4**
**Nameable Class**

**Table 2 Attributes of *Nameable***

| *Attribute Name* | *Description* | *Access* | *Reqd* | *Form* |
|---|---|---|---|---|
| *uid* | Identifier that uniquely and permanently identifies a *Nameable*. The *uid* contains a "uuid" string. See §4, ISO/IEC 11578:1996 for a reference to uuid definition. A properly formed uuid provides an identifier that is unique from any other uuid (in the factory and beyond). | RO | Y | String. |
| *name* | Human-readable name for *Nameable*. The *name* is representative of the role or function that the Nameable plays in the equipment structure. | RO | Y | String – See rules for naming below. |
| *description* | Human-readable description of *Nameable*. Contents of this attribute shall provide enough information for a human to understand the purpose of this *Nameable*. | RO | Y | String. |

8.5.3.1.1 *uid* — The requirements for each *uid* value are:

- The *uid* value shall conform to the rules for constructing a uuid as defined in standard ISO/IEC 11578:1996.
- The *uid* shall be a fixed value with respect to the physical hardware and shall be unique among all *Nameables*. The *uid* shall remain with the *Nameable*, even if the *Nameable* is removed from one equipment and added to another. For example, if an *IODevice* fails and is removed from an equipment, its replacement will have a different *uid*. If the original *IODevice* is repaired and installed on a different equipment, it will retain its original *uid*. Note that a *MaterialLocation* is contained within an equipment component and will move with that component, retaining its own *uid*.

8.5.3.1.2 *name* — The naming requirements for each *Nameable* are:

- *Names* shall consist of alphanumeric characters, spaces, hyphens, and underscores. Names shall start with an alpha character, and shall not contain leading or trailing spaces.
- The value of the *name* attribute shall be unique among component parts[1] of the same aggregate (object). This restriction holds for a *Nameable* instance that is aggregated (shared) by more than one object.
- If a *Nameable* represents a class or variable described in a SEMI standard, the *name* used shall conform to the spelling and case style specified in the standard. If the uniqueness rules can't be met, the *name* shall include an appended suffix of the form "-n", where 'n' is a number making the name unique.
- The *name* attribute represents the role or function of the object in the equipment hierarchy. The value of *name* shall not change unless the physical equipment configuration changes. Upon a configuration change, if a *Nameable* is replaced in its role and function by a different one, the *name* value representing that role shall be transferred to the new *Nameable*. For example, if an *IODevice* named "TemperatureControlZone5" fails and is replaced with a new unit, that new unit shall be named "TemperatureControlZone5".

8.5.3.2 *EquipmentElement* — This abstract class represents the basic information required for each hardware component in the equipment structure (see Figure 7). *EquipmentElement* includes the attributes: *elementType*, *supplier*, *make*, *model*, *modelRevision*, *function*, and *immutableId* (see Table 3). It also has a composite aggregation of *SoftwareModules*.

1 Members of an aggregation (i.e. the classes/objects opposite the diamond) are referred to as "component parts" or "components" of the aggregation. The "aggregate" is the class/object at the diamond end of the aggregation relationship.

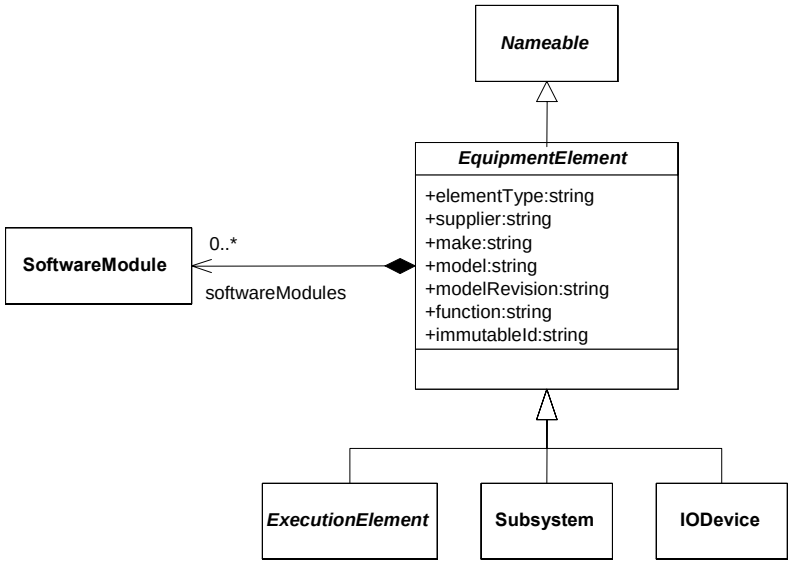



**Figure 5**
***EquipmentElement* Class**

**Table 3 Attributes of *EquipmentElement***

| *Attribute Name* | *Definition* | *Access* | *Reqd* | *Form* |
|---|---|---|---|---|
| elementType | Describes *EquipmentElement* type. For example, if *EquipmentElement* is a valve for controlling Nitrogen gas flow, the value of this attribute might be "Nitrogen Valve". If it is a load port, the value might be "Load Port". | RO | Y | String |
| supplier | The name of the manufacturer of the *EquipmentElement*, if known. Not all *EquipmentElement* instances of the equipment make it easy to determine this information automatically. Where this information cannot be determined or configured, it shall have the value "unknown". | RO | Y | String |
| make | The hardware make of the *EquipmentElement*, if known. Not all *EquipmentElement* instances of the equipment make it easy to determine this information automatically. Where this information cannot be determined or configured, it shall have the value "unknown". | RO | Y | String |
| model | The hardware model of the *EquipmentElement*, if known. Not all *EquipmentElement* instances of the equipment make it easy to determine this information automatically. Where this information cannot be determined or configured, it shall have the value "unknown". | RO | Y | String |
| modelRevision | The hardware model revision of the *EquipmentElement*, if known. Not all *EquipmentElement* instances of the equipment make it easy to determine this information automatically. Where this information cannot be determined or configured, it shall have the value "unknown". | RO | Y | String |
| function | The role of this *EquipmentElement* within the equipment. | RO | Y | String |
| immutableId | An unchanging identifier, such as the serial number of the *EquipmentElement*, if known. | RO | Y | String |

8.5.3.3 *ExecutionElement* — This abstract class models the parts of the equipment structure capable of processing, measuring, or testing material — *Module* and *Equipment*. It is a subclass of *EquipmentElement* (see Figure 8). *ExecutionElement* has the following attributes: *processName*, *processType*, and *recipeType* (defined in Table 4).

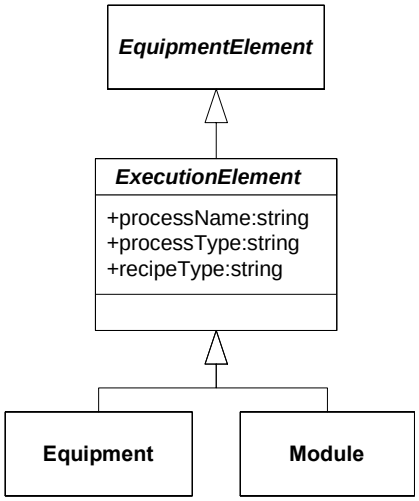



**Figure 6**
***ExecutionElement* Class**

**Table 4 Attributes of *ExecutionElement***

| *Attribute Name* | *Definition* | *Access* | *Reqd* | *Form* |
|---|---|---|---|---|
| processName | Describes the processing that occurs at this *ExecutionElement* (e.g., "coat", "develop", "bake", etc.). | RO | Y | String |
| processType | Categorizes the primary processing function of this *ExecutionElement* as an enumerated value chosen from the values: Measurement, Process, Storage, or Transport. See ¶8.5.3.3.1 for description of *processType* values. | RO | Y | Enumeration |
| recipeType | Instances of *ExecutionElement* having identical strings for *recipeType* are capable of executing the same set of recipes (e.g. processing instructions). Note that identical results are not guaranteed. | RO | Y | String |

8.5.3.3.1 The definition of *processType* lists four enumerated values. These are described as follows:

- *Measurement* — Refers to *Equipment* or *Modules* whose intended function is to measure or inspect the product and report results.
- *Process* — Refers to *Equipment* or *Modules* whose intended function is to process product by adding value to the product.
- *Storage* — Refers to *Equipment* or *Modules* whose intended function is primarily to provide storage, either short-term or long-term, for carriers (e.g. stockers).
- *Transport* — Refers to *Equipment* or *Modules* whose intended function is primarily to move material from one location to another. Transport equipment may also provide short-term storage for material.

8.5.3.4 *Extension* — This abstract class provides the ability for other SEMI standards and implementers to extend the CEM. *Extension* has no superclass and no attributes (see Table 5). It is associated with *Nameable* (see Figure 9). This association is navigable in both directions so that an *Extension* can identify the class that it modifies and a *Nameable* knows its *Extensions*.

8.5.3.4.1 *Extension* is modeled to allow extension of any *Nameable*. However, in practice, only concrete objects that inherit from *Nameable* shall have extensions.

8.5.3.4.2 As an abstract class, *Extension* must have a concrete subclass in order to be useful. The definition of subclasses of *Extension* is left to the implementer and/or other standards to define as extensions to the CEM. See ¶6.3 for a discussion of how the *Extension* class may be used to extend the Common Equipment Model.

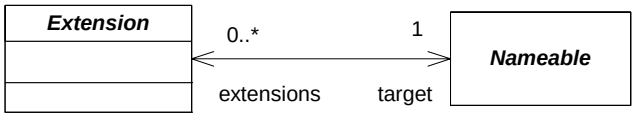



**Figure 7**
**Extension Class**

**Table 5 Attributes of Extension**

| *Attribute Name* | *Definition* | *Access* | *Reqd* | *Form* |
|---|---|---|---|---|
| | <<none defined>> | | | |

8.5.3.4.3 Since each class in the CEM inherits from *Nameable*, this extension method applies to the entire CEM. However, for ease of management, this specification limits the extensions. ***Extensions* shall be applied to concrete CEM classes only.**

8.5.3.4.4 Any number of subclasses of *Extension* may be created and thus be available as extensions of CEM classes. **No extension shall conflict with or override any feature of the CEM.** So, for example, no CEM defined attributes may be redefined as a different type.

8.5.4 *Concrete Classes*

8.5.4.1 *Equipment* — This class models the equipment as a whole. *Equipment* is a subclass of *ExecutionElement* (see Figure 10). An *Equipment* is an aggregate of *Modules*, *Subsystems*, and *IODevices*. It must contain a total of at least one of these components, but may contain zero or more of each. An *Equipment* that can communicate information about or act on behalf of other *Equipment* will also model associations with these *Equipment*[2]. *Equipment* defines no additional attributes defined in the Common Equipment Model (see Table 6).

8.5.4.1.1 Figure 10 shows that a *Module* or *Subsystem* may belong to more than one *Equipment*. The case where these components can be shared by multiple *Equipment* is unusual. An example of this would be where an in-line metrology tool is shared. In most cases, a *Module* or *Subsystem* will belong to zero or one *Equipment.*

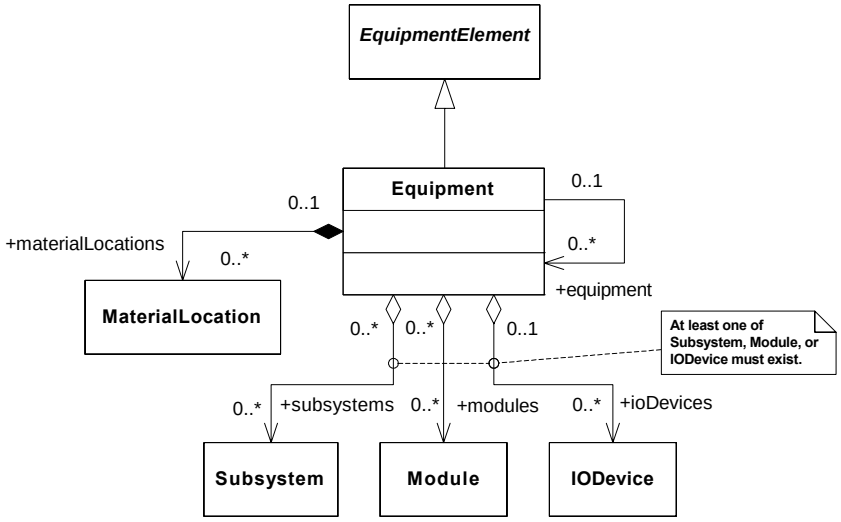



**Figure 8**
**Equipment Class**

2 This specification does not attempt to encourage or endorse any particular approach to structuring a single equipment or equipment acting in tandem. It attempts only to provide a means for representing any structure that could reasonably be expected to exist.

**Table 6 Attributes of Equipment**

| *Attribute Name* | *Definition* | *Access* | *Reqd* | *Form* |
|---|---|---|---|---|
| | <<none defined>> | | | |

8.5.4.2 *Module* — This class models major subsystems of the equipment, such as process chambers. *Module* is a subclass of *ExecutionElement* (see Figure 11).

8.5.4.2.1 A *Module* is capable of having (zero or more) *MaterialLocations* and is capable of processing material. A *Module* or one of its aggregated *Subsystem* or *Module* objects must contain at least one *MaterialLocation*.

8.5.4.2.2 A *Module* must be a component part of *Equipment* or another *Module* and may be shared by multiples of these. It may also be an aggregation of *Subsystem*, *IODevice*, and/or other *Module* instances (zero or more each).

8.5.4.2.3 *Module* has no additional attributes defined in the Common Equipment Model (see Table 7).

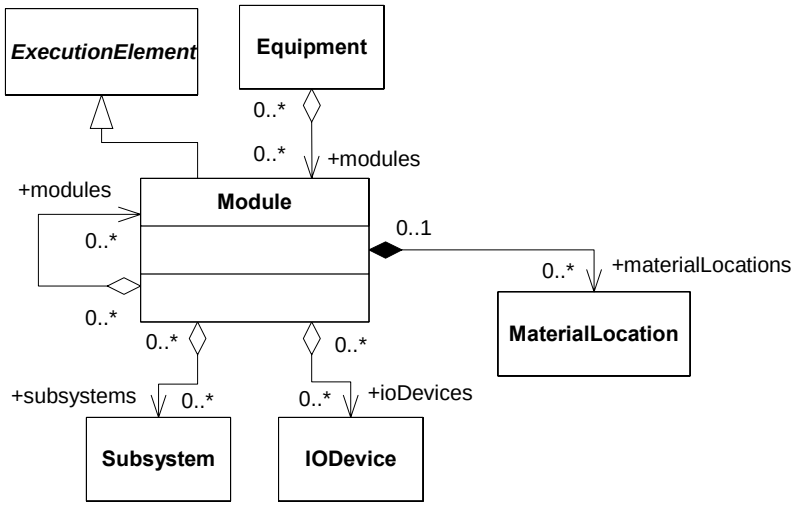



**Figure 9**
**Module Class**

**Table 7 Attributes of Module**

| *Attribute Name* | *Definition* | *Access* | *Reqd* | *Form* |
|---|---|---|---|---|
| | <<none defined>> | | | |

8.5.4.3 *Subsystem* — This class models subsystem and subassembly components of the equipment (see Figure 12). *Subsystem* is a subclass of *EquipmentElement*.

8.5.4.3.1 A *Subsystem* may be capable of having (zero or more) *MaterialLocations*, but is not capable of processing material.

8.5.4.3.2 A *Subsystem* must be a component part of *Equipment*, *Module*, or another *Subsystem* and may be shared by multiples of these. It may also be an aggregation of *Subsystem*, or *IODevice* instances (zero or more each).

8.5.4.3.3 *Subsystem* has no additional attributes defined in the Common Equipment Model (see Table 8).

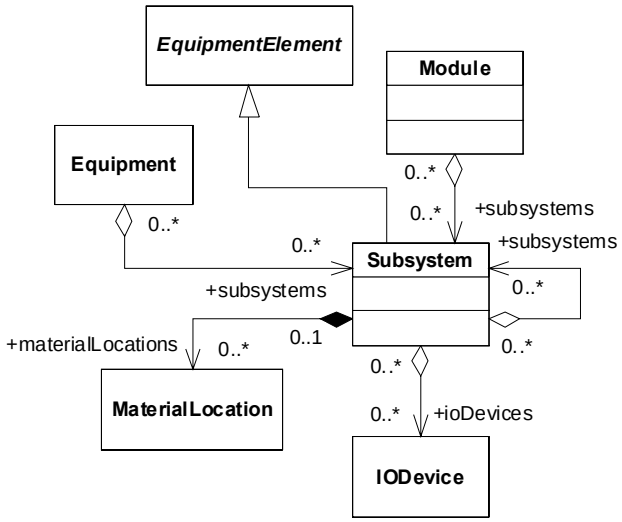



**Figure 10**
**Subsystem Class**

**Table 8 Attributes of Subsystem**

| *Attribute Name* | *Definition* | *Access* | *Reqd* | *Form* |
|---|---|---|---|---|
| | <<none defined>> | | | |

8.5.4.4 *IODevice* — This class models sensors, actuators, or intelligent actuator/sensor devices. Suppliers shall model this *EquipmentElement* on equipment if it is practical and useful to do so for the purposes of process control and technical support. *IODevice* is a subclass of *EquipmentElement* and is a component part of zero or more *Equipment*, *Subsystem*, and/or *Module* instances (see Figure 13). *IODevice* has no additional attributes defined in the Common Equipment Model (see Table 9).

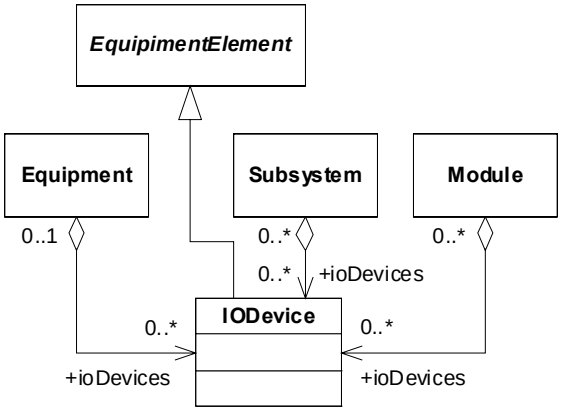



**Figure 11**
**IODevice Class**

**Table 9 Attributes of IODevice**

| *Attribute Name* | *Definition* | *Access* | *Reqd* | *Form* |
|---|---|---|---|---|
| | <<none defined>> | | | |

8.5.4.5 *MaterialLocation* — This concrete class models the ability of certain equipment components to hold material. *MaterialLocation* is a subclass of *Nameable*. The attributes of *MaterialLocation* class are defined in Table 10.

8.5.4.5.1 A *MaterialLocation* is exclusively related to a single *Equipment*, *Module*, or *Subsystem* in a composite aggregation (see Figure 14). Each *Equipment*, *Module*, and *Subsystem* may have zero or more *MaterialLocations*. *Equipment*, *Modules*, *and Subsystems* may also have indirect access to *MaterialLocations* if they aggregate a component that has a *MaterialLocation*. So, for instance, a *Module* may not have a *MaterialLocation*, but may contain a *Subsystem* with a *MaterialLocation*.

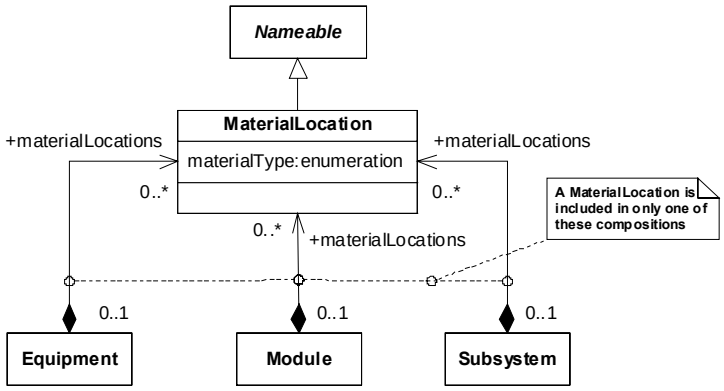



**Figure 12**
**MaterialLocation Class**

**Table 10 Attributes of MaterialLocation**

| *Attribute Name* | *Definition* | *Access* | *Reqd* | *Form* |
|---|---|---|---|---|
| *materialType* | Type of material held. Enumeration value is one of the following: Carrier, Substrate, ProcessDurable | RO | Y | Enumeration |

8.5.4.6 *SoftwareModule* — This concrete class is used to describe the existence and version of equipment system software that is in use on the equipment and on equipment components. Such software might be supplied by the equipment manufacturer or by third parties. It includes any software that may be changed without affecting the identity of the hardware. This would cover software stored on disk drives, flash memory, eeproms, and other memory devices. This class allows the factory host to track changes in the software. See ¶8.5.3.2 *EquipmentElement* for the definition of the modelRevision attribute that tracks changes in hardware. Tracking hardware and software revisions together allows better detection of changes in the equipment capability.

8.5.4.6.1 The *SoftwareModule* class is a component of *EquipmentElement* in a composite aggregation (see Figure 15). It has four attributes: *name*, *supplier*, *description*, and *version* (see Table 11). In the case where a *SoftwareModule* affects multiple *EquipmentElements*, each would contain its own separate *SoftwareModule* describing the same software.

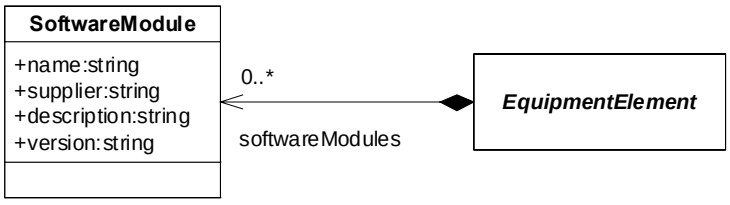



**Figure 13**
**SoftwareModule Class**

**Table 11 Attributes of SoftwareModule**

| *Attribute Name* | *Definition* | *Required* | *Form* |
|---|---|---|---|
| *name* | The name of the SoftwareModule. | Y | String |
| *supplier* | The name of the company that created the SoftwareModule. Where this information cannot be determined or configured, it shall have the value “unknown”. | Y | String |
| *description* | A description of the function or purpose of the SoftwareModule. | Y | String |
| *version* | The version number of the SoftwareModule. It should be formatted according to the version numbering convention of the company that produced the SoftwareModule. | Y | String |

## 9 Positional Referencing Of *Nameable* Objects

9.1 *Introduction*

9.1.1 The E120 specification provides fixed referencing of all *Nameable* objects by means of the *uid* attribute. The *uid* is fixed to a specific hardware entity. Consider a massflow controller that fails and is swapped out with one of the same kind. It is expected that the *name* attribute of that massflow controller would become the same as the old one, but it is required that the *uid* be different. Thus, the *name* is distinctively a “positional reference” that identifies a place or position in the hierarchy.

9.1.2 This section defines a positional reference that is unique within the equipment. This “Locator” value is based on the *name* attribute of *Nameable*. It can be used in communications to the equipment to uniquely identify a Nameable object in that equipment.

9.1.3 If the equipment provides a positional reference for an equipment component, the Locator form, as defined in this section, shall be used.

9.1.4 There are some situations where a positional reference to an equipment component is more useful than the fixed reference that *uid* provides. For example, a data collection plan might indicate that during processing, the reading from a massflow controller (one kind of *Nameable*) is to be recorded. At some point, the massflow controller is replaced. If the reference in the specification is to the *uid*, then the data collection plan no longer works. In a worse case, two identical massflow controllers are removed from similar process modules for maintenance and then replaced on the opposite modules. In this case, the *uid* reference would yield data from the wrong process module. If a positional reference were used, then the data collection plan would always collect readings from the massflow controller on the specified process module.

9.2 *Constructing A Locator*

9.2.1 Values for the Locator string are derived by starting with the *name* attribute of the root *Nameable* (usually an Equipment) in the hierarchy followed by the ‘/’ character. This is followed by the *names* of any objects between the root and target (each *name* separated by the ‘/’ character). Finally, the *name* of the target is added. A shorthand notation for this pattern is: “root/aggregate/aggregate/aggregate/target”. Example values for Locator are illustrated by the block diagram in Figure 14.

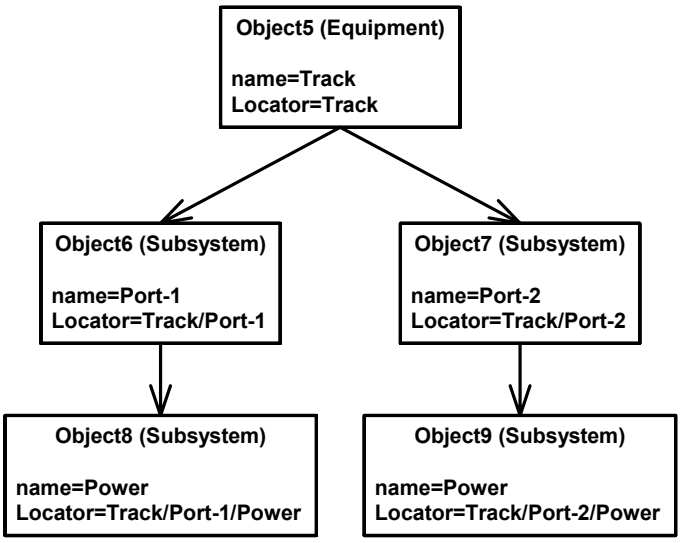



**Figure 14**
**Example: Equipment Hierarchy**

9.2.2 In the case where an object is shared, there may be more than one Locator that will resolve to that object. The block diagram in Figure 15 gives an example of the Locators that might result. Notice that Object8 is shared by Object6 and Object7. This means that it can be referenced by two different but equally valid Locator values as shown in the diagram. For *Nameable* instances with more than one possible Locator value, any valid Locator construct will resolve to that object.

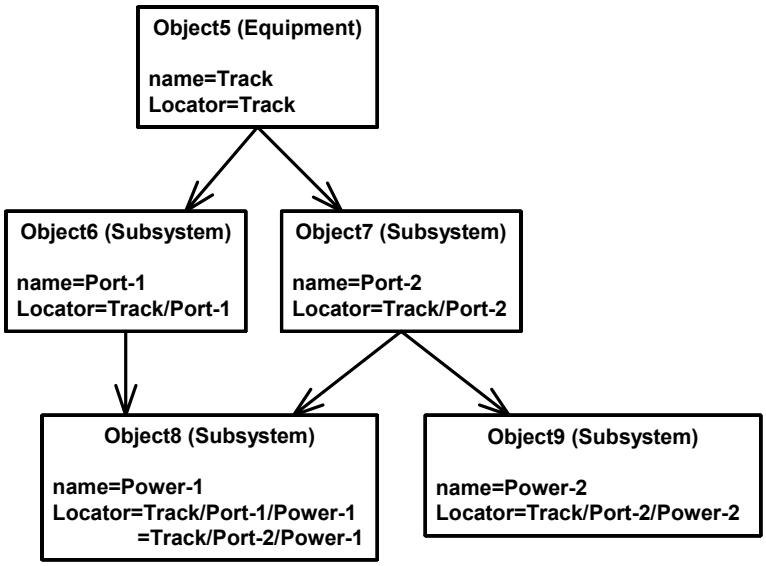



**Figure 15**
**Example: Equipment Hierarchy With Shared Object**

9.2.3 Note that the Locator value is valuable as a reference to a *Nameable*. However, that value is not available from the *Nameable* as an attribute. The Locator can be built independently and used when only the *names* are known or when the position in the hierarchy is more important than the identity of the equipment component being referenced.

## 10 Compliance

10.1 This section defines requirements for compliance to this specification.

10.1.1 The term "CEM class" refers generally to any one of the classes defined by this specification. The term "equipment component" refers generally to any part of the equipment that fits the definition of any subclass of *EquipmentElement* (including *Equipment*, *Module*, *Subsystem*, and *IODevice*).

10.1.2 Compliance to this specification requires

1. Satisfaction of all requirements listed in Table 12, and
2. Satisfaction of all requirements for each CEM class that has been implemented (as reflected in the corresponding row of Table 13).

10.1.3 The tables below each contain columns labeled "Implemented" and "CEM Compliant". Each row represents a requirement (or group of requirements).

10.1.4 The "Implemented" column communicates an assertion that the equipment has provided the function or capability that corresponds to the intent of that requirement. This assertion is not testable. However, in order to be "CEM Compliant", the "Implemented" column would logically be set to "yes".

10.1.5 A row/requirement is considered to be "CEM Compliant" when all defined aspects of that requirement have been satisfied. Therefore, an implementation that is compliant to CEM would have:

- All rows of Table 12 marked "yes" for both "Implemented" and "CEM Compliant",
- The "Equipment" row of Table 13 would be marked "yes" for both "Implemented" and "CEM Compliant", and
- Any other rows of Table 13 that are marked "yes" for "Implemented" would also be marked "yes" for CEM Compliant".

10.2 *General CEM Compliance Requirements*

10.2.1 There are five general CEM compliance requirements. They are listed in Table 12.

**Table 12 General CEM Requirement Compliance Table**

| # | *CEM Requirement* | *Implemented* | *CEM Compliant* |
|---|---|---|---|
| 1 | An *Equipment* instance shall be defined using the CEM *Equipment* class | ☐ Yes ☐ No | ☐ Yes ☐ No |
| 2 | When other equipment component instances are defined, they shall meet all requirements for the corresponding CEM classes (see Table 13 and §8). Please note that this specification does not require that all CEM classes be used in each equipment implementation. Instead, it requires that all requirements for a class be met whenever it is used. | ☐ Yes ☐ No | ☐ Yes ☐ No |
| 3 | All object definitions that have been based on CEM classes shall be made available electronically to the factory host. | ☐ Yes ☐ No | ☐ Yes ☐ No |
| 4 | If the equipment provides a positional reference for an equipment component, the Locator form as defined in §9 shall be used. | ☐ Yes ☐ No | ☐ Yes ☐ No |
| 5 | Association Navigability shall be supported as defined in ¶6.2.7. | ☐ Yes ☐ No | ☐ Yes ☐ No |

10.3 *Specific Compliance Requirements*

10.3.1 Table 13 lists each concrete class and references the section where requirements are defined for that class. The requirements are primarily defined by the UML diagram and table of attributes in the referenced section. The text of the section serves to clarify the requirements and add detail where appropriate.

10.3.2 On the UML diagram, note the associations and their cardinality and role specifications. Associations (including aggregations) constitute requirements on a given class if navigable to the target class. (See ¶6.2.7 for an explanation about Navigability and related requirements). For a navigable association, object(s) of the target class are required. When the cardinality of the target class of the association may be zero, the association and target class may be omitted as appropriate.

10.3.3 In Table 13, the first column represents a CEM concrete class. "CEM Compliance" in this table indicates (1) that at least one instance of that class exists and (2) that all instances of that class meet all requirements for the class.

10.3.4 The "Class/Superclass Definition" column gives a section number reference to each class in the inheritance hierarchy for the concrete class in that row. The requirements for all classes in the inheritance hierarchy apply for this concrete class.

10.3.5 The "Navigable Via Association" column lists the section number for the definition of each class that may be associated with the concrete class in that row. When such an association exists, there must be a CEM Compliant instance of that associated class. If the cardinality of the association may be zero, then the existence of the association is dependent on the equipment structure.

**Table 13 CEM Class Compliance Table**

| *CEM Class Requirement* | *Class/Superclass Definition* | *Navigable Via Association* | *Implemented* | *CEM Compliant* |
|---|---|---|---|---|
| *Equipment* | 8.5.3.1 Nameable<br>8.5.3.2 EquipmentElement<br>8.5.3.3 ExecutionElement<br>8.5.4.1 Equipment | 8.5.3.4 Extension<br>8.5.4.2 Module<br>8.5.4.3 Subsystem<br>8.5.4.4 IODevice<br>8.5.4.5 MaterialLocation<br>8.5.4.6 SoftwareModule | ☐ Yes ☐ No | ☐ Yes ☐ No |
| *Module* | 8.5.3.1 Nameable<br>8.5.3.2 EquipmentElement<br>8.5.3.3 ExecutionElement<br>8.5.4.2 Module | 8.5.3.4 Extension<br>8.5.4.2 Module<br>8.5.4.3 Subsystem<br>8.5.4.4 IODevice<br>8.5.4.5 MaterialLocation<br>8.5.4.6 SoftwareModule | ☐ Yes ☐ No | ☐ Yes ☐ No |
| *Subsystem* | 8.5.3.1 Nameable<br>8.5.3.2 EquipmentElement<br>8.5.4.3 Subsystem | 8.5.3.4 Extension<br>8.5.4.3 Subsystem<br>8.5.4.4 IODevice<br>8.5.4.5 MaterialLocation<br>8.5.4.6 SoftwareModule | ☐ Yes ☐ No | ☐ Yes ☐ No |
| *IODevice* | 8.5.3.1 Nameable<br>8.5.3.2 EquipmentElement<br>8.5.4.4 IODevice | 8.5.3.4 Extension<br>8.5.4.6 SoftwareModule | ☐ Yes ☐ No | ☐ Yes ☐ No |
| *MaterialLocation* | 8.5.3.1 Nameable<br>8.5.4.5 MaterialLocation | 8.5.3.4 Extension | ☐ Yes ☐ No | ☐ Yes ☐ No |

10.4 *Compliance Verification Approach*

10.4.1 Verification of compliance to SEMI E120 may be accomplished by comparing the objects and their attributes that are reported from the equipment with their class definitions from CEM and with the other requirements listed in this Table 12.

10.4.2 Electronic verification requires that all CEM related objects be collected directly from the equipment along with all their attributes. The method for collection will vary depending on the type of electronic communication available. Since the CEM classes and their attributes are static, the method and time of collection shall not affect the verification outcome.

10.4.3 Once the objects are collected, the set is checked electronically against the requirements defined in Table 12 and Table 13 above. The process will ensure that the names of all classes and attributes are correct, that any required objects exist and that each instance provides all defined attributes and appropriate associations.

## 11 Related Documents

11.1 Unified Modeling Language (UML) Specification, Version 1.4, OMG Specification 01-09-67, available from http://www.omg.org/technology/documents/modeling_spec_catalog.htm.[3]

3 This specification uses object-modeling terminology consistent with this OMG standard for UML.

# RELATED INFORMATION 1
# FURTHER BACKGROUND AND JUSTIFICATION

**NOTICE:** This Related Information is not an official part of SEMI E120. This Related Information is not intended to modify or supersede the official standard. Determination of the suitability of the material is solely the responsibility of the user.

## R1-1 UML Terminology

R1-1.1 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.

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

R1-1.2 These definitions are taken from B.2 Glossary of Terms of the above document.

R1-1.2.1 *abstract class* — a class that cannot be directly instantiated. Contrast: *concrete class*.

R1-1.2.2 *aggregate* [class] — a class that represents the "whole" in an aggregation (whole-part) relationship. See: *aggregation.*

R1-1.2.3 *aggregation* — a special form of association that specifies a whole-part relationship between the aggregate (whole) and a component part. See: *composition*.

R1-1.2.4 *association* — the semantic relationship between two or more classifiers that specifies connections among their instances.

R1-1.2.5 *attribute* — a feature within a classifier that describes a range of values that instances of the classifier may hold.

R1-1.2.6 *cardinality* — the number of elements in a set. Contrast: *multiplicity*.

R1-1.2.7 *child* — in a generalization relationship, the specialization of another element, the parent. See: *subclass*, *subtype*. Contrast: *parent*.

R1-1.2.8 *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*.

R1-1.2.9 *classifier* — a mechanism that describes behavioral and structural features. Classifiers include interfaces, classes, datatypes, and components.

R1-1.2.10 *classification* — the assignment of an object to a classifier. See *dynamic classification*, *multiple classification* and *static classification*.

R1-1.2.11 *class diagram* — a diagram that shows a collection of declarative (static) model elements, such as classes, types, and their contents and relationships.

R1-1.2.12 *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*.

R1-1.2.13 *concrete class* — a class that can be directly instantiated. Contrast: *abstract class.*

R1-1.2.14 *constraint* — a semantic condition or restriction. Certain constraints are predefined in the UML, others may be user defined.

R1-1.2.14.1 Constraints are one of three extensibility mechanisms in UML. See: *tagged value, stereotype.*

R1-1.2.15 *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.

R1-1.2.16 *element* — an atomic constituent of a model.

R1-1.2.17 *feature* — a property, like operation or attribute, which is encapsulated within a classifier, such as an interface, a class, or a datatype.

R1-1.2.18 *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.*

R1-1.2.19 *inheritance* — the mechanism by which more specific elements incorporate structure and behavior of more general elements related by behavior. See: *generalization.*

R1-1.2.20 *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.*

R1-1.2.21 *interface* — a named set of operations that characterize the behavior of an element.

R1-1.2.22 *interface inheritance* — the inheritance of the interface of a more general element. Does not include inheritance of the implementation. Contrast: *implementation inheritance*.

R1-1.2.23 *method* — the implementation of an operation. It specifies the algorithm or procedure associated with an operation.

R1-1.2.24 *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 to as simply the model.

R1-1.2.25 *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*.

R1-1.2.26 *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.*

R1-1.2.27 *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.*

R1-1.2.28 *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.

R1-1.2.29 *parent* — in a generalization relationship, the generalization of another element, the child. See: *subclass, subtype*. Contrast: *child.*

R1-1.2.30 *qualifier* — an association attribute or tuple of attributes whose values partition the set of objects related to an object across an association.

R1-1.2.31 *relationship* — a semantic connection among model elements. Examples of relationships include associations and generalizations.

R1-1.2.32 *role* — the named specific behavior of an entity participating in a particular context. A role may be static (e.g., an association end) or dynamic (e.g., a collaboration role).

R1-1.2.33 *signature* — the name and parameters of a behavioral feature. A signature may include an optional returned parameter.

R1-1.2.34 *subclass* — in a generalization relationship, the specialization of another class; the superclass. See: *generalization*. Contrast: *superclass*.

R1-1.2.35 *subtype* — in a generalization relationship, the specialization of another type; the supertype. See: *generalization*. Contrast: *supertype*.

R1-1.2.36 *superclass* — in a generalization relationship, the generalization of another class; the subclass. See: *generalization.* Contrast: *subclass*.

R1-1.2.37 *supertype* — in a generalization relationship, the generalization of another type; the subtype. See: *generalization*. Contrast: *subtype*.

R1-1.2.38 *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 also: *implementation class*. Contrast: *interface*.

# SEMI E120.1-0705
# XML SCHEMA FOR THE COMMON EQUIPMENT MODEL (CEM)

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

## Table of Contents



## List of Figures

# List of Tables



## 1 Purpose

1.1 The purpose of this specification is to provide an XML schema that corresponds to the UML model for equipment defined by SEMI E120. This schema is available to equipment communication standards that want to use CEM-defined equipment structural information in XML formatted equipment communication.

## 2 Scope

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

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

## 3 Referenced Standards and Documents

3.1 *SEMI Standards*

SEMI E120 — Provisional Specification for the Common Equipment Model (CEM)

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

3.2 *OMG Standards*

Unified Modeling Language (UML) Specification[1], Version 1.4, OMG Specification 01-09-67

3.3 *W3C Standards*

Extensible Markup Language (XML) 1.0 (Second Edition)[2] — W3C, 6 October 2000

Namespaces in XML[3] — W3C, 14 January 1999

XML Schema Part 0: Primer[4] — W3C, 2 May 2001

XML Schema Part 1: Structures[5] — W3C, 2 May 2001

XML Schema Part 2: Datatypes[6] — W3C, 2 May 2001

---

1 Object Management Group, Inc., 250 First Ave. Suite 100, Needham, MA 02494, U.S.A., Ph: 1-781-444 0404, Fax: 1-781-444 0320, http://www.omg.org/technology/documents/modeling_spec_catalog.htm

2 World Wide Web Consortium (W3C), http://www.w3.org/TR/2000/REC-xml-20001006/

3 World Wide Web Consortium (W3C), http://www.w3.org/TR/1999/REC-xml-names-19990114/

4 World Wide Web Consortium (W3C), http://www/w3/org/TR/xmlschema-0/

5 World Wide Web Consortium (W3C), http://www/w3/org/TR/xmlschema-1/

6 World Wide Web Consortium (W3C), http://www/w3/org/TR/xmlschema-2/

XML Path Language (Xpath)[7] — W3C, 16 November 1999

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

## 4 Terminology

4.1 *Abbreviations and Acronyms*

4.1.1 *UML* — Unified Modeling Language

4.1.2 *W3C* — World Wide Web Consortium

4.1.3 *XML* — eXtensible Markup Language

4.2 *Definitions and Acronyms*

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

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

## 5 Conventions

5.1 This section discusses the conventions used in this specification for translating UML to XML and for documenting the XML. These conventions are heavily influenced by the SEMI E121.

5.2 The reader is expected to have a working knowledge of the UML, XML, and Schema specifications (See ¶3.2 and ¶3.3). This document does not provide tutorial information on these subjects.

5.3 *Translating UML to XML*

5.4 This document follows the guidelines for XML as outlined in SEMI E121.

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

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

5.6 The translation of a UML class to XML is documented using a table format illustrated Table 1. Note that SEMI E120 does not define services, so only attribute and association representations are addressed here. Any superclass information is addressed in the text and the diagrams.

5.7 *Translation Table Column Header Description*

- *Attribute or Role Name* — If an attribute, the name of the attribute is placed here. If an association (including aggregation or composition), the role name from the UML diagram is placed here. Compositions are often not assigned roll names. In that case “none” is placed here.
- *UML Name/Type* — If an attribute, the data type of the UML attribute is place here. If an association, the type of association is placed here. The possible types are “Composition”, “Aggregation”, or the basic “Association”. UML defines these three types of association.
- *XML Element or Attribute* — Lists the type of XML construct used to represent the UML attribute or association.

7 World Wide Web Consortium (W3C), http://www/w3/org/TR/xpath/

- *XML Name/Type* — Provides the name and data type of the resulting XML construct. The type may be a built-in type (for example, xs:string), or a named type defined within the XML schema.

**Table 1 Example Translation Table**

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

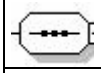
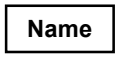
5.8 *Documenting The XML With Diagrams*

5.8.1 This document provides graphical representations of the included XML. Although no standard graphical notation for XML could be found, various XML tools have their own notation. This document will use the notation provided by XMLSPY® from Altova Corporation[8]. There is no requirement for use of XMLSPY® in order to comply with this specification. Figure 1 shows a sample XML diagram that will be used to provide a basis for explanation of the XML graphical notation used in the rest of the document.

5.8.2 In the diagram, rectangular boxes represent XML elements. Ownership or containment is read from right to left in the diagrams. In the sample diagram, ParentType contains Child1, Child2, Child3, and Child4. In turn, Child2 contains Child2a, Child2b, and Child2c. The additional symbols (8-sided boxes) represent sequences or choices. See Table 2 for an explanation of these symbols.

5.8.3 Please note that “element”, “sequence”, “all”, and “choice” are XML terms (see the W3C standards referenced in ¶3.3). The reader is expected to be familiar with XML.

**Table 2 Altova XMLSPY Schema Diagram Symbols**

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

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

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

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

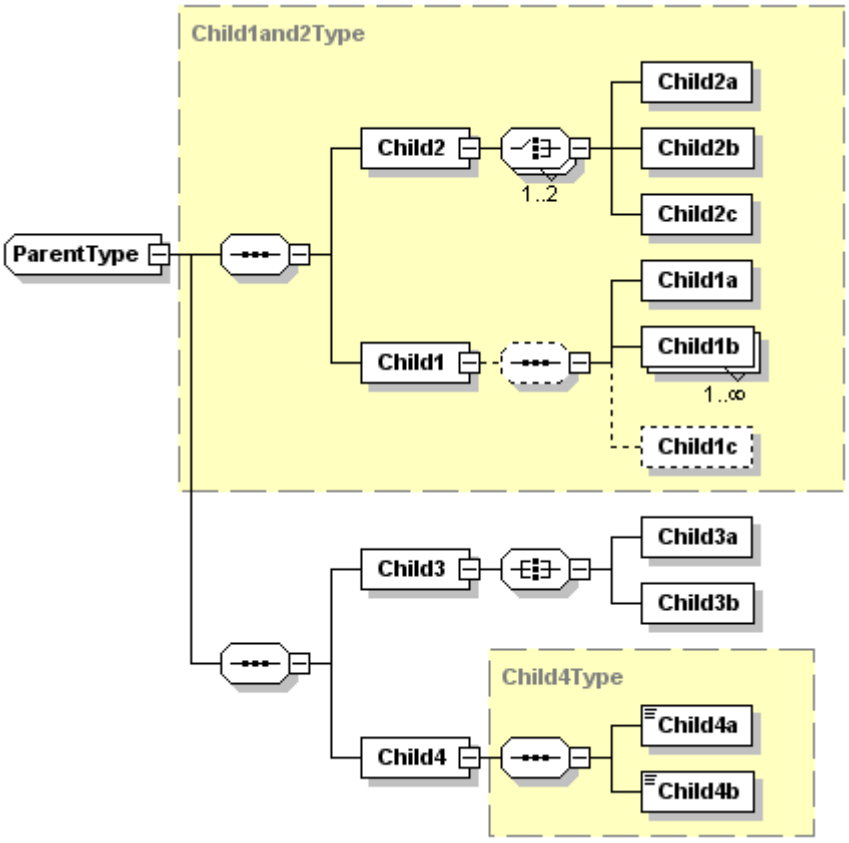



**Figure 1**
**XML Example Diagram**

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

5.8.7 Reading Figure 1 would yield the following additional information:

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

5.8.8 *XML Sample*

5.8.8.1 The sample XML for the example shown in Figure 1, is presented below. Refer to the XML documentation referenced in ¶3.3for a complete description of the syntax and semantics of XML.

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

**Figure 2**
**XML for Sample**

## 6 CEM Class Mapping to XML

6.1 This section describes how the CEM classes from SEMI E120 were mapped to XML. The descriptions in this section are provided to support and explain the XML schema document.

6.2 *Abstract Classes*

6.2.1 The classes from the CEM that were designated "abstract" are included in this section. Abstract classes are not directly implemented. They exist in the model (and in the schema) to provide a basis for the concrete objects. So, they will not have corresponding instances in an XML instance file based on this schema.

6.2.2 *Nameable => NameableType*

6.2.2.1 The Nameable class is mapped to the XML NameableType. The XML definition of the class Nameable uses simple types UuidType and NameType (see ¶6.4.2 for definition).

6.2.2.2 The association of Nameable with the class Extension is modeled here as an array (Extensions) contained inside of Nameable. See ¶6.2.5 for more about Extension.

**Table 3 Translation Table For Nameable**

| *Attribute or Role Name* | *UML Type* | *XML Element or Attribute (or Reference)* | *XML Name/Type* |
|---|---|---|---|
| uid | string | element | Uid: UuidType |
| name | string | element | Name: NameType |
| description | string | element | Description: xs:string |
| extensions | association | element | Extensions: ExtensionArrayType |

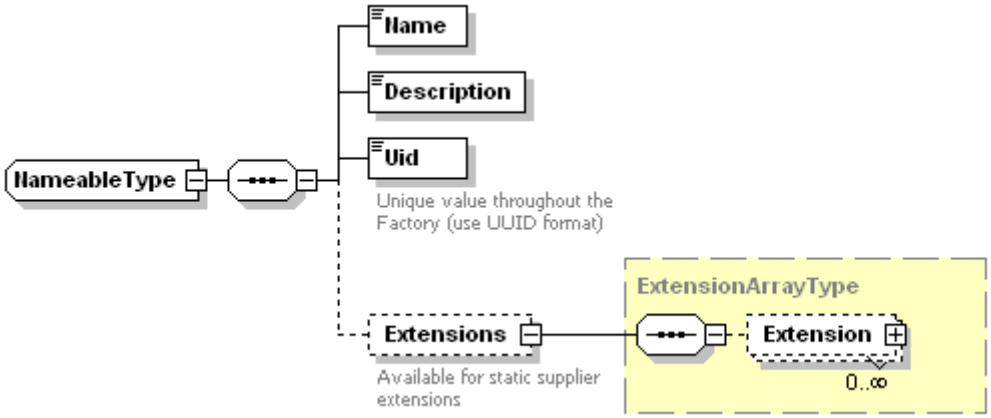



**Figure 3**
**NameableType**

6.2.3 *EquipmentElement => EquipmentElementType*

6.2.3.1 The EquipmentElement class is mapped to the XML EquipmentElementType. EquipmentElement translates in a straightforward way with all UML attributes mapping to elements. EquipmentElementType extends NameableType.

**Table 4 Translation Table For EquipmentElement**

| *Attribute or Role Name* | *UML Type* | *XML Element or Attribute (or Reference)* | *XML Name/Type* |
|---|---|---|---|
| elementType | string | element | ElementType: xs:string |
| supplier | string | element | Supplier: xs:string |
| make | string | element | Make: xs:string |
| model | string | element | Model: xs:string |
| modelRevision | string | element | ModelRevision: xs:string |
| function | string | element | Function: xs:string |
| immutableId | string | element | ImmutableId: xs:string |

| *Attribute or Role Name* | *UML Type* | *XML Element or Attribute (or Reference)* | *XML Name/Type* |
|---|---|---|---|
| softwareModules | composition | element | SoftwareModules: SoftwareModuleArrayType |

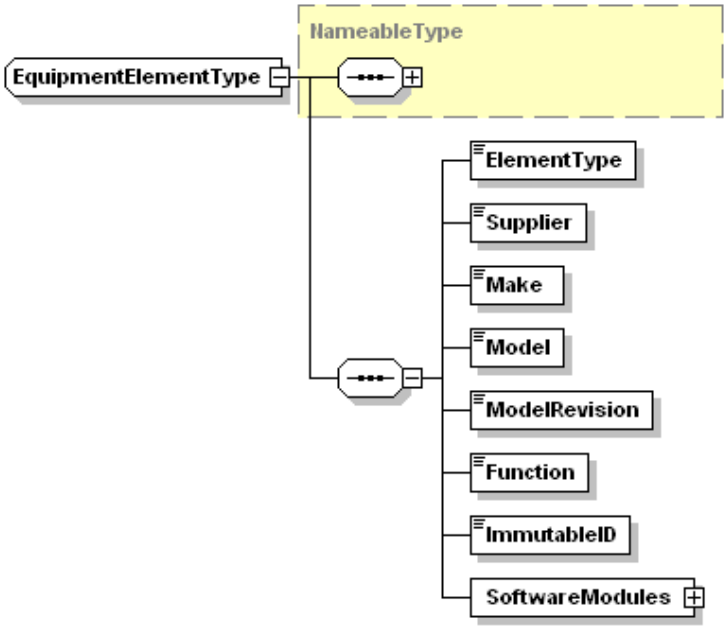



**Figure 4**
**EquipmentElementType**

6.2.4 *ExecutionElement => ExecutionElementType*

6.2.4.1 ExecutionElement class maps to the XML ExecutionElementType. ExecutionElement's three attributes map directly to XML elements. A special type "ProcessTypeEnum" has been created to represent the enumeration processType. ExecutionElementType extends EquipmentElementType.

**Table 5 Translation Table For ExecutionElement**

| *Attribute or Role Name* | *UML Type* | *XML Element or Attribute (or Reference)* | *XML Name/Type* |
|---|---|---|---|
| processName | string | element | ProcessName: xs:string |
| processType | enumeration | element | ProcessType: ProcessTypeEnum |
| recipeType | string | element | RecipeType: xs:string |

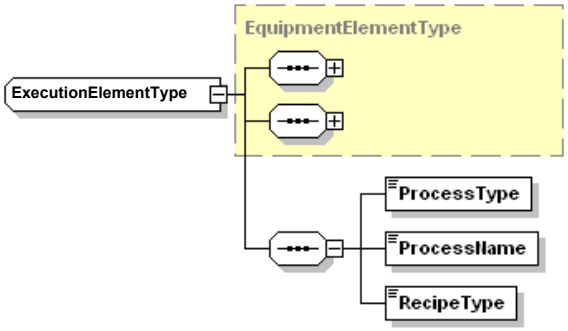



**Figure 5**
**ExecutionElement Type**

6.2.5 *Extension => ExtensionType*

6.2.5.1 The Extension class maps to the XML ExtensionType. Extension class has no attributes, but it has a navigable association to the Nameable class. This association is represented as the element NameableRef which contains the uid attribute of the associated Nameable.

6.2.5.2 The SEMI E120 specification says that CEM classes are extended by creating subclasses of the Extension class. In this XML schema, this mechanism is represented by the addition of xs:any to ExtensionType. The XML rendering of the additions by the UML subclass of Extension are placed directly in the ExtensionType in this way.

**Table 6 Translation Table For Extension**

| *Attribute or Role Name* | *UML Type* | *XML Element or Attribute (or Reference)* | *XML Name/Type* |
|---|---|---|---|
| target | association | element | NameableRef: UuidType |

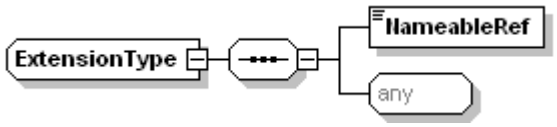



**Figure 6**
**ExtensionType**

6.3 *Concrete Classes*

6.3.1 This section contains the XML mappings for the classes defined in SEMI E120 as concrete classes. These are the classes that can be instantiated an XML instance document based on this schema.

6.3.2 The mapping from UML to XML was created with the goal of having the simplest XML instance documents for most equipment. It was assumed that cases of shared equipment components (Modules, Subsystems, or IODevices belonging to more than one aggregation) are rare and can be treated as special cases. Where no components are shared, the simplest solution is to map all aggregated objects to be contained within the object that represents the aggregate. In XML, this translates to an Element that contains a set of sequences containing the instances of the components—one sequence for each type of component. This is the way one would normally map a composite aggregation.

6.3.3 In order to deal with the occasional shared component, a second set of sequences was added for each type of component that can be shared. These sequences contain references (via the uid attribute) to the shared components that are defined elsewhere. So, if a Subsystem is shared by two Modules, then the Subsystem instance is defined once and included inside one of the Module instances. In the other Module instance, this Subsystem appears in the sequence of references to Subsystems and is identified by the uid of the original.

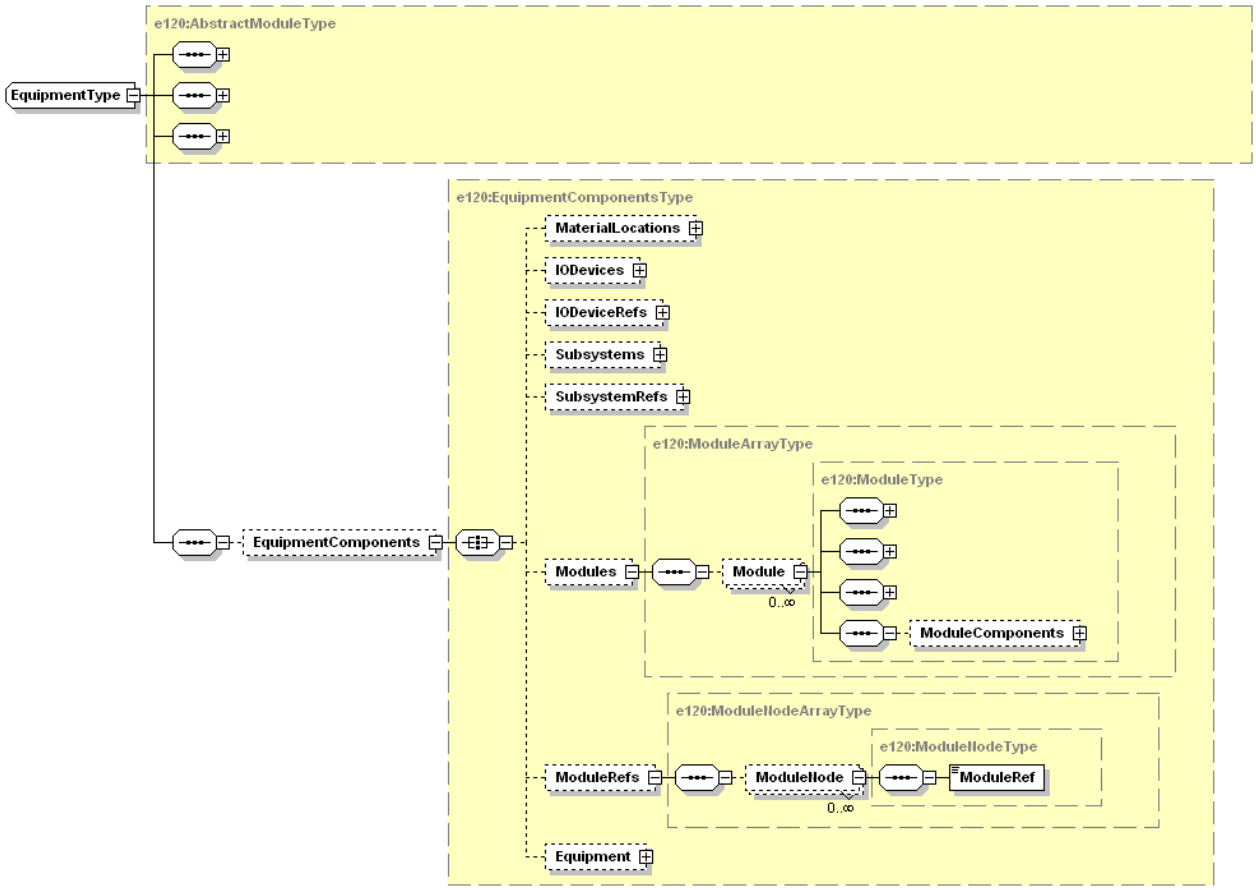



**Figure 7**
**Example Of The XML Structure For Shared Components**

6.3.4 In summary, with this approach, the structure of the equipment is readily apparent in the XML instance document except for the components that are shared. When a component is shared, its definition does not need to be repeated. Instead, a reference by uid is provided. This avoids additional complexity and duplication that could bloat the XML instance document.

6.3.5 *Equipment => EquipmentType*

6.3.5.1 The Equipment class is mapped to the XML EquipmentType. EquipmentType extends the ExecutionElementType (see Figure 8). It is also a container for all of the components of the Equipment – Modules, Subsystems, IODevices, MaterialLocations, and other Equipment.

6.3.5.2 Table 7 shows how the associations of the Equipment class are mapped into XML. The EquipmentType contains the element EquipmentComponents to act as a container. The XML elements listed in the table are contained in EquipmentComponents.

**Table 7 Translation Table For Equipment**

| *Attribute or Role Name* | *UML Type* | *XML Element or Attribute (or Reference)* | *XML Name/Type* |
|---|---|---|---|
| equipment | aggregation | element | Equipment: EquipmentArrayType |
| modules | aggregation | element (Module instances) | Modules: ModuleArrayType |
| | | element (reference – when shared) | ModuleRefs: ModuleRefArrayType |
| subsystems | aggregation | element (Subsystem instances) | Subsystems: SubsystemArrayType |
| | | element (references – when shared) | SubsystemRefs: SubsystemRefArrayType |

| *Attribute or Role Name* | *UML Type* | *XML Element or Attribute (or Reference)* | *XML Name/Type* |
|---|---|---|---|
| iodevices | aggregation | element (IODevice instances) | IODevices: IODeviceArrayType |
| | | element (references – when shared) | IODeviceRefs: IODeviceRefArrayType |
| materialLocations | composition | element MaterialLocations | MaterialLocations: MaterialLocationArrayType |

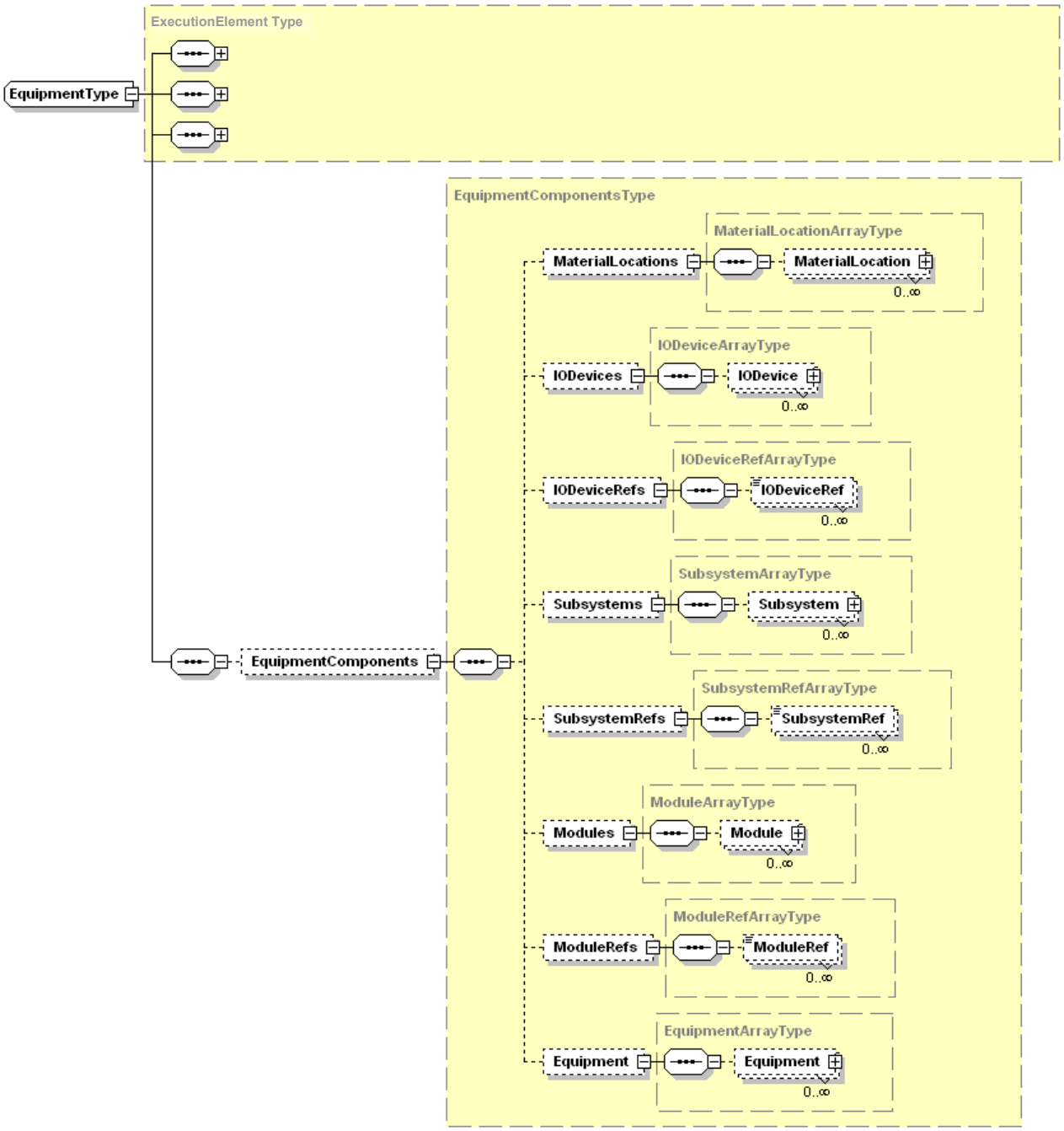



**Figure 8**
**EquipmentType**

6.3.6 *Module => ModuleType*

6.3.6.1 The Module Class is mapped to the XML ModuleType. The ModuleType extends the ExecutionElementType (see Figure 9). It is also a container for all of the components of the Module – Subsystems, IODevices, MaterialLocations, and other Modules.

6.3.6.2 Table 8 shows how the attributes and associations of the Module class are mapped into XML. The ModuleType contains the element ModuleComponents to act as a container. The XML elements listed in the table are contained in ModuleComponents.

**Table 8 Translation Table For Module Class**

| *Attribute or Role Name* | *UML Type* | *XML Element or Attribute (or Reference)* | *XML Name/Type* |
|---|---|---|---|
| modules | aggregation | element (used first time a Module is defined in the system) | Modules: ModuleArrayType |
| | | element (references a previously defined Module) | ModuleRefs: ModuleRefArrayType |
| subsystems | aggregation | element (used first time a Subsystem is defined in the system) | Subsystems: SubsystemArrayType |
| | | element (references a previously defined Subsystem) | SubsystemRefs: SubsystemRefArrayType |
| iodevices | aggregation | element (used first time an IODevice is defined in the system) | IODevices: IODeviceArrayType |
| | | element (references a previously defined IODevice) | IODeviceRefs: IODeviceRefArrayType |
| materialLocations | composition | element | MaterialLocations: MaterialLocationArrayType |

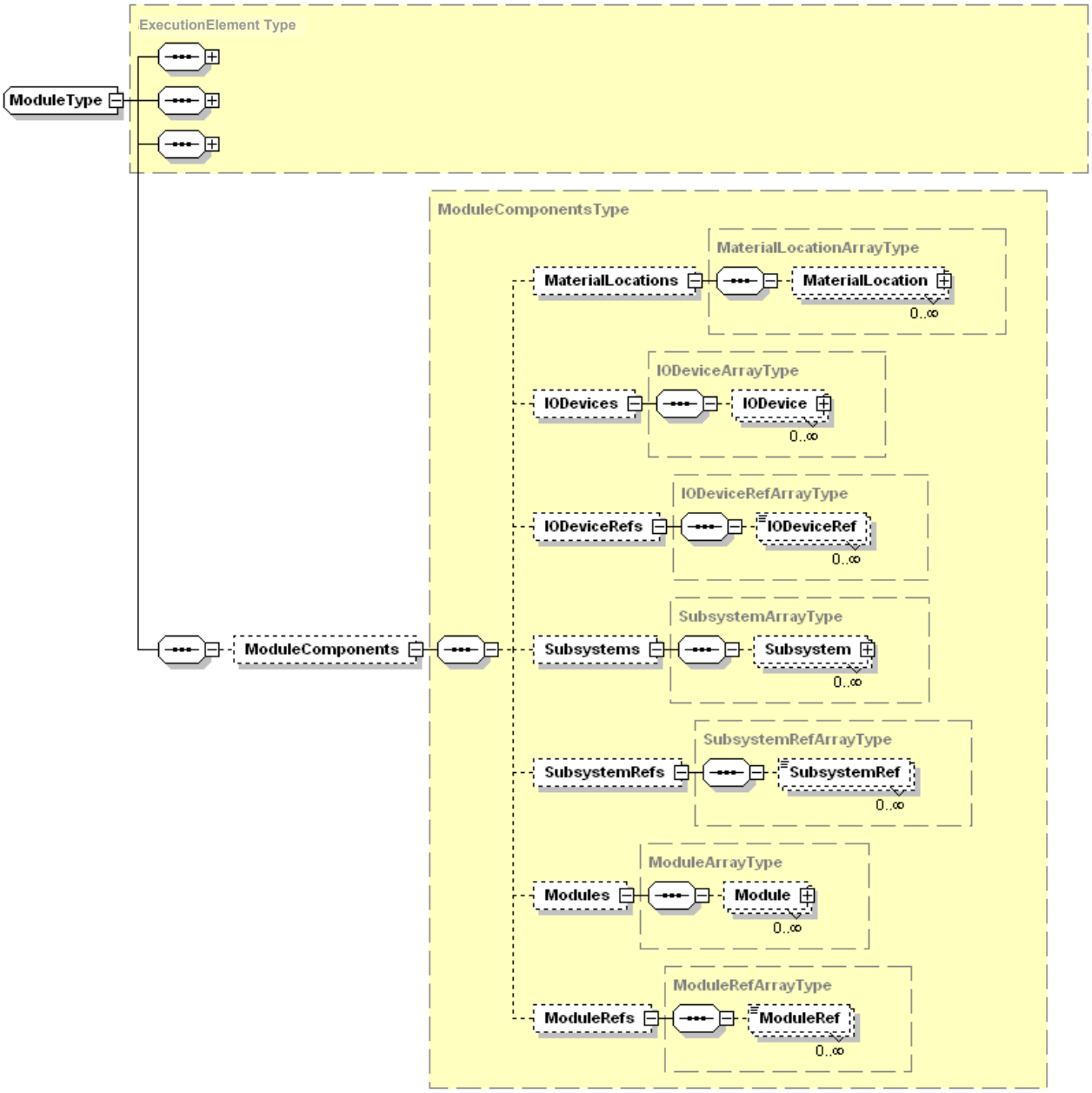



**Figure 9**
**ModuleType**

6.3.7 *Subsystem => SubsystemType*

6.3.7.1 The Subsystem class is mapped to the XML SubsystemType. SubsystemType extends EquipmentElement (see Figure 10). It is also a container for all of the components of the Subsystem – IODevices and other Subsystems.

6.3.7.2 Table 9 shows how the associations of the Subsystem class are mapped into XML. The SubsystemType contains the element SubsystemComponents to act as a container. The XML elements listed in the table are contained in SubsystemComponents.

**Table 9 Translation Table For Subsystem**

| *Attribute or Role Name* | *UML Type* | *XML Element or Attribute (or Reference)* | *XML Name/Type* |
|---|---|---|---|
| subsystems | aggregation | element (used first time a Subsystem is defined in the system) | Subsystems: SubsystemArrayType |
| | | element (references a previously defined Subsystem) | SubsystemRefs: SubsystemRefArrayType |
| iodevices | aggregation | element (used first time an IODevice is defined in the system) | IODevices: IODeviceArrayType |
| | | element (references a previously defined IODevice) | IODeviceNodeArrayType |
| materialLocations | composition | element | MaterialLocations: MaterialLocationArrayType |

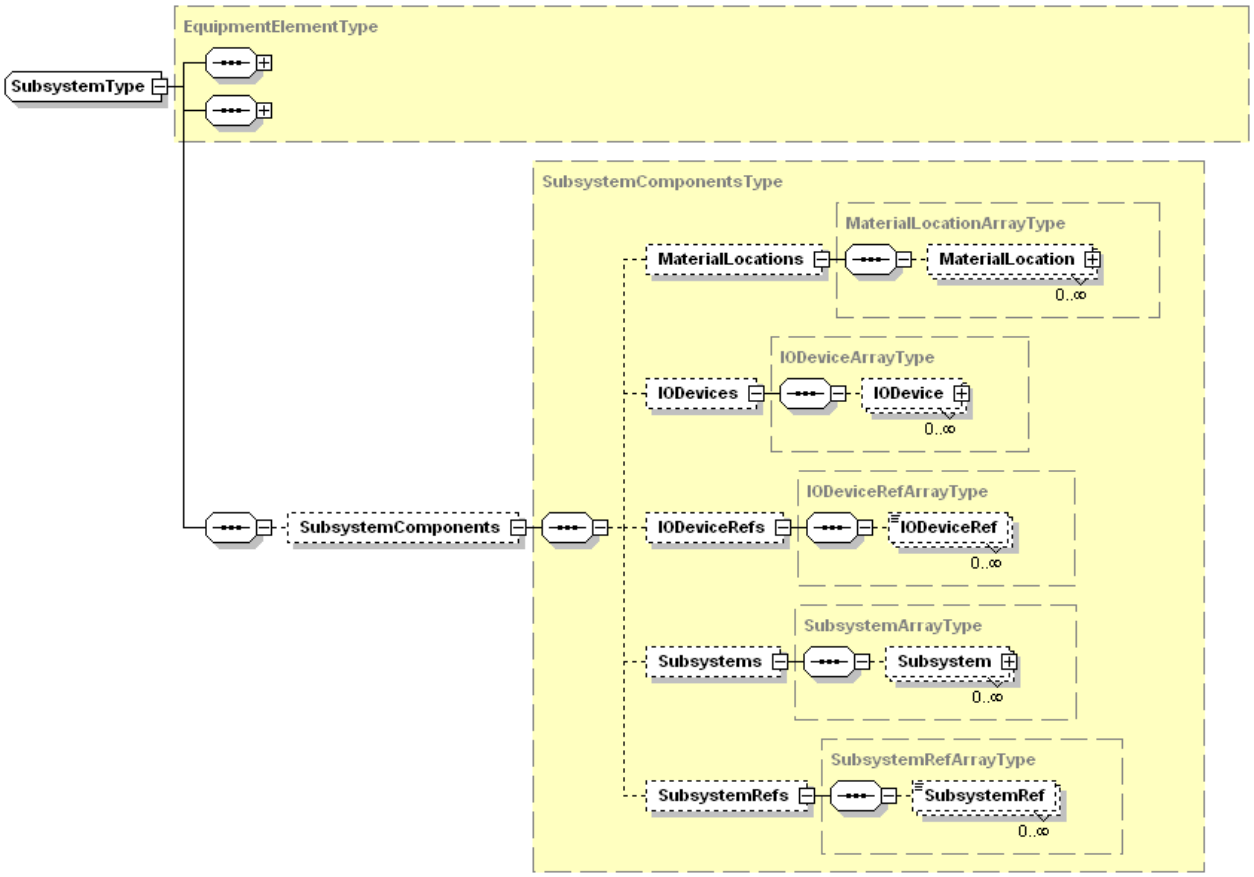



**Figure 10**
**SubsystemType**

6.3.8 *IODevice => IODeviceType*

6.3.8.1 The IODevice class is mapped to the XML IODeviceType. IODeviceType defines no attributes or navigable associations, but it does extend EquipmentElementType (see Figure 11). It is included in multiple associations, but it cannot navigate any of them. Therefore, its translation table is empty.

**Table 10 Translation Table For IODevice**

| *Attribute or Role Name* | *UML Type* | *XML Element or Attribute (or Reference)* | *XML Name/Type* |
|---|---|---|---|
| none | | | |