R1-1.3.2 *Requesting A Table* — Table R1-4 illustrates a client's request for a certain table.

Table R1-4 Requesting a Table

	Comments	*Client*	*Dir*	*IMM*	*Comments*
1		TableRequest.req	→		
		ObjSpec = object specifier for the IMM, TableType = "TableIMM_RawData", TableID = "P19343445-Lot103102#5", TableCmd = Entire table;			
			←	TableRequest.req	
				ObjSpec = object specifier for the IMM, TableType = "TableIMM_RawData", TableID = "P19343445-Lot103102#5", list of TableAttr = name/value attribute pairs; list of ColHdr = list of names of column headers; list of values for 50 rows, TableAck = Success	

R1-1.3.3 *Automatic Table Send* — Table R1-5 illustrates the messaging for AutoTableSend.

Table R1-5 Automatic Table Send

	Comments	*Client*	*Dir*	*IMM*	*Comments*
1		SetAutoTableSend.req	→		
		AutoSendRawData = False; AutoSendConvertedData = True,, AutoSendCalibrationData = False			
			←	SetAutoTableSend.rsp	
					Converted Data Table "abc" completes.
			←	Event.nfy	IMMDTSM: T4
			←	TableSend.req	
				ObjSpec = null string, TableType = "TableIMM_ConvertedData", TableID = "abc", TableCmd = Entire table; list of TableAttr = name/value attribute pairs; list of ColHdr = list of names of column headers; list of values for 50 rows	
		TableSend.rsp	→		
		TableAck = success			

R1-1.4 *Typical IMM Data Table Object Sequence of Events*

R1-1.4.1 Table R1-6 shows a typical sequence of event messages that occur during normal substrate processing. In this example, an IMM Raw Data Table Object is created as soon as processing (measuring) begins. As soon as the first row of this table is completed, an IMM Converted Data Table Object is created.

R1-1.4.2 The controlling process job may be complete prior to the completion of the IMM Raw Data Table Object, as soon as the measured substrate has been unloaded by the STPO. This is not shown.

R1-1.4.3 DataTime values in the example below indicate sequence only. Actual date/time values are needed for implementations.

Table R1-6 Typical Flow of Events for IMM Data Table Objects

	Comments	*Client*	*Dir*	*IMM*	*Comments*
1			←	Event.nfy	Process job “ABC” starts.
				PrJobID = “ABC” SubstrID = “XYZ” DateTime = 0000	Check on right prjob id name.
			←	Event.nfy	Data table created – IMMDTSM-T1.
				DateTime = 0001 TableType = “TableIMM_RawData” TableID = “FR100” RecipeID = “Reflection33-LEVEL5” SubsrID = “XYZ”	
2a			←	Event.nfy	Row complete – IMMDTSM-T2.
				DateTime = 0002 TableType = “TableIMM_RawData” TableID = “FR100” RecipeID = “Reflection33-LEVEL5” SubsrID = “XYZ” RowNum = 1	
			←	Event.nfy	IMM Converted Data Table created – IMMDTSM-T1.
				DateTime = 0003 TableType = “TableIMM_ConvertedData” TableID = “FT100” RecipeID = “FilmThickness-LEVEL5” SubsrID = “XYZ”	
2b			←	Event.nfy	The above message is sent as rows i = 2-8 complete.
				DateTime = 0004…0010 TableType = “TableIMM_RawData” TableID = “FR100” RecipeID = “Reflection33-LEVEL5” SubsrID = “XYZ” RowNum = i	
			←	Event.nfy	Raw data table completed – IMMDTS M-T4.

	Comments	*Client*	*Dir*	*IMM*	*Comments*
				DateTime = 0004…0011 TableType = “TableIMM_RawData” TableID = “FR100” RecipeID = “Reflection33-LEVEL5” SubsrID = “XYZ”	
3			←	Event.nfy	1st row of converted data completed – IMMDTSM-T2.
				DateTime = 0012 TableType = “TableIMM_ConvertedData” TableID = “FT100” RecipeID = “FilmThickness-LEVEL5” SubsrID = “XYZ” RowNum = 1	
4			←	Event.nfy	Row complete – IMMDTSM-T2, for row i = 2–8.
				DateTime = 0013 - 0019 TableType = “TableIMM_ConvertedData” TableID = “FT100” RecipeID = “FilmThickness-LEVEL5” SubsrID = “XYZ” RowNum = i	
3b			←	Event.nfy	Table completed – IMMDTSM-T4.
				DateTime = 0020 TableType = “TableIMM_ConvertedData” TableID = “FT100” RecipeID = “FilmThickness-LEVEL5” SubsrID = “XYZ” SummaryData =xxx	

RELATED INFORMATION 2
USE OF IMMC WITH EPT

NOTICE: This related information is not an official part of SEMI E127 and was derived from work of the originating committee. This related information was approved for publication by full letter ballot procedures.

R2-1 Association of IMMC State Changes with EPT Module State Changes

R2-1.1 The IMMC defines the IMM as an EPT Module, and the tool connected to the IMM is required to include the IMM among its EPT Modules.

R2-1.2 The EPT state of the IMM as a whole is based on three components: the state of the STPO, the state of the portion of the IMM that is independent of the STPO (representing loading and unloading of substrates) and the combined state of any IMM Data Table Object (representing either current measurement (raw data) or calibration) or data conversion activities. The IMMC does not require the IMM-only (exclusive of STPO and IMM Data Table Object states) to have a state model representing any other activities that it may be performing, but to correctly calculate the EPT state as a whole, its own EPT states must be done.

R2-1.2.1 *IMM-Only State* — Excluding activities performed by, and subsystems used only by, an STPO or represented by an IMM Data Table Object state, the IMM-only may contain various subsystems, such as an aligner, the various sensors used for measuring, and mechanisms for moving a substrate from one Substrate Location to another, but excludes those mechanisms used solely by an STPO. The IMM-only, then, will be either:

- actively performing some task, with or without material, with no faults (EPT-BUSY),
- inactive, without material, and with no reason that would prevent it from performing any of its normal tasks (EPT-IDLE), or
- inactive, either with material present or because of a fault in one of its subsystems (EPT-BLOCKED).

R2-1.2.2 The IMM Service state is not related to the IMM EPT State. Whether IN SERVICE or OUT OF SERVICE, the IMM-Only can be in any of the three EPT states.

R2-1.3 *STPO State*

R2-1.3.1 The STPO is IDLE only in the READY TO LOAD state. Here it has no material, is performing no activity, and has nothing to prevent it from performing a load activity.

R2-1.3.2 The STPO is BUSY only in the ASSIGNED state.

R2-1.3.3 The STPO is BLOCKED in the BLOCKED state.

R2-1.3.4 The STPO is BLOCKED in the READY TO UNLOAD state, because, although it is not performing an activity, it has material.

R2-1.4 *IMM Data Table Object State*

R2-1.4.1 The combined IMM Data Table Object state, for EPT purposes, is determined by the combination of all IMM Data Table Objects stored within the IMM.

R2-1.4.2 The combined state is BUSY if at least one IMM Data Table Object is IN PROCESS.

R2-1.4.3 If no IMM Data Table Object is IN PROCESS, then the combined state is IDLE.

R2-1.4.4 There is no state for an IMM Data Table Object that corresponds to BLOCKED.

R2-1.5 The rules for determining the IMM EPT State are as follows:

- If any of the above components is BLOCKED, then the IMM EPT State is BLOCKED.
- If no component is BLOCKED and at least one is BUSY, then the IMM EPT State is BUSY.
- If all three components are IDLE, then the IMM EPT State is IDLE.

R2-1.6 The statements in the above sections are summarized in tabular format by Table R2-1. It illustrates the fact that BLOCKED has precedent over BUSY and IDLE, and BUSY has precedence over IDLE.

Table R2-1 Determination of IMM EPT State

IMM ONLY State	*STPO State*	*IMM Data Table State*	*IMM EPT State*
IDLE	READY TO LOAD	all IN RETENTION	IDLE
BLOCKED	any	any	BLOCKED
BUSY	any but BLOCKED	any	BUSY
any	BLOCKED	any	BLOCKED
any but BLOCKED	ASSIGNED	any	BUSY
any but BLOCKED	any but BLOCKED	one or more IN PROCESS	BUSY

SEMI E127.1-0705
SPECIFICATION FOR SECS-II PROTOCOL FOR INTEGRATED MEASUREMENT MODULE COMMUNICATIONS (IMMC)

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

1 Purpose

1.1 This document maps the services and data of IMMC to SECS-II streams and functions and data definitions.

2 Scope

2.1 This document applies to all implementations of IMMC that use the SECS-II message protocol (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 Referenced Standards and Documents

3.1 *SEMI Standards*

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

SEMI E39 — Object Services Standard: Concepts, Behavior, and Services

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

SEMI E90.1 — Provisional Specification for SECS-II Protocol Substrate Tracking

SEMI E127 — Specification for Integrated Measurement Module Communications: Concepts, Behavior, and Services (IMMC)

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

4 Conventions

4.1 Names of SECS-II data items are in all upper case. Names of SEMI E127 services, parameters, and attributes are in Title or mixed case.

4.2 SML notation is used for SECS-II format codes. See SEMI E30, Section 8.4.

5 Services Mapping

5.1 Table 1 shows the specific SECS-II streams and functions that shall be used for SECS-II implementations of the services defined in IMMC.

Table 1 Services Mapping Table

Message Name	*Stream, Function*	*SECS-II Message Name*
ChangeService	S14, F19/20	Generic Service Request/Acknowledge
ClientConnect	S14, F19/20	Generic Service Request/Acknowledge
ClientDisconnect	S14, F19/20	Generic Service Request/Acknowledge
RequestRetentionConditions	S14, F19/20	Generic Service Request/Acknowledge
SetAutoTableSend	S14, F19/20	Generic Service Request/Acknowledge
SetRetentionConditions	S14, F19/20	Generic Service Request/Acknowledge
TransferPathCalibration	S14, F19/20	Generic Service Request/Acknowledge

Message Name	*Stream, Function*	*SECS-II Message Name*
XfrCmdListReq	S14, F19/20	Generic Service Request/Acknowledge
XfrCommand.req	S4, F29	Handoff Command (HO)
XfrCommand.rsp	S4, F31	Handoff Command Complete (HCC)
XfrVerify.req	S4, F33	Handoff Verified (HV)
XfrVerify.rsp	S4, F33	Handoff Verified (HV)
XfrHalt.req	S4, F39	Handoff Halt (HH)
XfrHalt.rsp	S4, F41	Handoff Halt Acknowledge (HHA)

[#1] All Stream 4 messages are primaries with no replies defined. This avoids transaction timeouts.

[#2] The response to S4, F33 is another S4, F33.

[#3] Mappings for UpdateSubstrateObject are defined in SEMI E90.1. Mappings for Delete, GetAttr and SetAttr are defined in SEMI E39.1.

6 Service Parameter Mapping

6.1 Two tables providing mappings service parameter mappings are provided, one for those services mapped to S14, F19/20 and one for those services mapped to Stream 4.

6.2 Table 2 shows the mapping for service parameters and their elements for services mapped to S14, F19/20. The data item SVCNAME is the text string representation of the service name found in Table 1 and should be assumed as case-sensitive. The name of each parameter is the text string representation of the parameter name as it appears in Table 2.

Table 2 Service Parameter Mapping Table for S14, F19/20

Parameter Name (SPNAME)	*Parameter Range (SPVAL)*	*SECS-II Data Item Reference*	*SECS-II Format*
AutoSendCalibrationData	TRUE/FALSE	SPVAL	BOOLEAN
AutoSendConvertedData	TRUE/FALSE	SPVAL	BOOLEAN
AutoSendRawData	TRUE/FALSE	SPVAL	BOOLEAN
ClientID	Text	SPVAL	A[1,40]
ClientInterfaceVersion	Text	SPVAL	A[1,80]
ClientConnectAck	TRUE/FALSE	SVCACK	BOOLEAN
ClientConnectStatus	—		L, 2 1. <SVCACK> 2. <Status>
ClientType	Enumerated: “Ctrl” or “Data”	SPVAL	A[4]
Deltas	—	SPVAL	L,3 1. <F8> 2. <F8> 3. <F8>
ErrorCode	Enumerated	ERRCODE	U2
ErrorText	80 characters	ERRTEXT	A[1,80]
OrientationIn	Any	SPVAL	F8
OrientationOut	Any	SPVAL	F8
Properties	—	SPVAL	L, n 1. L, 2 1. <ATTRID 2. <ATTRDATA> . n. L,2 1. <ATTRID 2. <ATTRDATA>

Parameter Name (SPNAME)	*Parameter Range (SPVAL)*	*SECS-II Data Item Reference*	*SECS-II Format*
RetentionRules	—	SPVAL	L,n 1. <RetentionRule$_1$> . n. <RetentionRule$_n$>
ServiceAck	Enumerated	SVCACK	BIN
Status	—	SPVAL	L, n 1. L, 2 1. < ErrorCode > 2. < ErrorText > . n. L,2 1. < ErrorCode > 2. < ErrorText >
Substid	1–80 characters	SPVAL	[A1,80]
XfrCmdList	—	SPVAL	L,n 1. <HOCMDNAME$_1$> . n. <HOCMDNAME$_n$>

6.3 Table 3 provides the parameter mappings for services mapped to Stream 4.

Table 3 Service Parameter Mapping Table for Stream 4

Parameter Name	*Parameter Range*	*SECS-II Data Item Reference*	*SECS-II Format*
ErrorCode	Enumerated	ERRCODE	U2
ErrorText	80 characters	ERRTEXT	A[1,80]
ObjID names/value	Text = “ObjID” Text = value of STPO ObjID	CPNAME$_1$ CPVAL$_1$	A[5] A[1,80]
Status	—	SPVAL	L, n 1. L, 2 1. <ErrorCode> 2. <ErrorText> . n. L,2 1. < ErrorCode > 2. < ErrorText >
XfrAck	TRUE/FALSE	HOACK	BOOLEAN
XfrCmdName	1–20 characters	HOCMDNAME	A[1,20]
XfrHaltAccept	0–1	HOHALTACK	I1
XfrLink	0–65,535	TRLINK	U2
XfrStatus	—	—	L, 2 1. <HOACK> 2. <Status>

6.4 Table 4 shows the data items in SECS II messages that do not have a corresponding service parameter.

Table 4 Additional Data Item Requirement Table

Function	*SECS-II Message*	*SECS-II Data Item*
Used to satisfy SECS-II conventions for linking a multi-block inquiry with subsequent multi-block message.	S14,F19	DATAID
Specifies the number of the HOCMDNAME in XfrCmdList, Table 2.	S4. F29	MCINDEX
Name of IMMC service. Enumerated text strings: ChangeService ClientConnect ClientDisconnect RequestRetentionConditions SetAutoTableSend SetRetentionConditions TransferPathCalibration XfrCmdListReq	S14, F19	SVCNAME
A unique number generated by the service requestor to identify a specific operation and connect it to a later completion confirmation. If there are multiple STPOs, OPID in the S14, F19 message used for UpdateSubstrateObject and TRLINK shall be assigned the same value to allow the hand-off messages in Stream 4 to be matched to the properties of the substrate being transferred.	S14, F19	OPID
Used to uniquely identify one substrate handoff from another. Assigned by the Control Client. If there are multiple STPOs, OPID in the S14, F19 message used for UpdateSubstrateObject and TRLINK shall be assigned the same value to allow the hand-off messages in Stream 4 to be matched to the properties of the substrate being transferred.	S4, F29, 31, 33, 39	TRLINK

6.5 Table 5 provides the parameter mapping for IMMC service parameters not specified in Tables 2 or 3.

Table 5 Mapping for Elements of Service Parameters

ElementName	*Mapping*	*SPVAL Format*
RetentionRule	Each RetentionRule is either a list of RetentionRules to be ANDed together, or a single RetentionRule. Each RetentionRule is sent as a name/value pair.	L,2 1. <RetentionRuleName> 2. <RetentionRuleValue> or L, n 1. <RetentionRule> . n. <RetentionRule>
RetentionRuleName	Enumerated text strings: “AfterXfr” “ClientDel” “MaxTbl” “RetTime”	A[6,9]
RetentionRuleValue	Any non-negative integer.	U2

6.6 Table 6 provides the mappings from ErrorCodes in SEMI E127 to ERRCODE values in SEMI E5.

Table 6 ErrorCode Mappings to ERRCODE Values

Service	*Valid Logical Values for ErrorCode*	*ERRCODE*
ClientConnect	Client Already Connected Duplicate ClientID Invalid ClientType IncompatibleVersions	32772 = Client Already Connected 32773 = Duplicate ClientID 32774 = Invalid ClientType 32775 = IncompatibleVersions
ClientDisconnect	Unrecognized ClientID (Client not currently connected)	32776 = Unrecognized ClientID (Client not currently connected)
XfrCommand	Rejected/Unrecognized Command Rejected/Parameter Error	32771 = Invalid command 47 – Invalid Parameter
XfrCommand XfrVerify	Failed (Completed Unsuccessfully) Failed (Unsafe) – External intervention required	32777 = Failed (Completed Unsuccessfully) 32778 = Failed (Unsafe) – External intervention required
XfrVerify	Sensor-Detected Obstacle Material Not Sent Material Not Received Material Lost Hardware Failure Transfer Cancelled	32779 = Sensor-Detected Obstacle 32780 = Material Not Sent 32781 = Material Not Received 32782 = Material Lost 32783 = Hardware Failure 32784 = Transfer Cancelled
All other services	Action performed – no errors Action will be performed at earliest opportunity Action can not be performed now Action failed due to errors Unrecognized command At least one parameter has an error	0 - No error. 32768 = Action will be performed at earliest opportunity 32769 = Action can not be performed now 32770 = Action failed due to errors 32771 = Invalid command 47 – Invalid Parameter

7 Object Attribute Form

7.1 §7 defines the SECS-II form of the attributes of objects defined in IMMC, presented in alphabetical order. Note that attribute names in this section indicate the literal text string that shall be accepted in string comparisons for attribute names in Stream 14 messages.

7.2 By convention, all tables include ObjID as well as ObjType as the two attributes required by SEMI E39. Table 7 specifies the attributes of the Integrated Measurement Module.

Table 7 Integrated Measurement Module Object Attribute Specification

Attribute Name	*Range/Value*	*SECS-II Format*
ObjType	"IMM"	A[3]
ObjID	Text. Conforms to restrictions for OBJID in SEMI E39.1, §6.	A[1,80]
EventList	List of 0+ text/integer pairs	L,n 1. L,2 1. <A[1,80] 2. <U2 or U4> . n. L,2 1. <A[1,80] 2. <U2 or U4>

Attribute Name	*Range/Value*	*SECS-II Format*
RetentionRules	List of enumerated text	L, n 1. <A[1,80]> . . n. <A[1,80]>
Service	Enumerated: 0 = OUT OF SERVICE 1 = IN SERVICE	I1
SubstrateList	List of 0+ OBJID	L, n 1. <A[1,80] > . . n. <A[1,80]>
TableCapacity	0–65,535	U2
TableCount	0–65,535	U2
TableStorageAlert	--	I(), F8

7.3 *Substrate Transfer Path Object* — Table 8 specifies the elements of the Substrate Transfer Path Object.

Table 8 Substrate Transfer Path Object Attribute Specification

Attribute Name	*Range/Value*	*SECS-II Format*
ObjType	Text = “SubstrateTransferPath”.	A[21]
ObjID	Text. Conforms to restrictions for OBJID in SEMI E39.1, §6.	A[1,80]
EventList	List of 0+ text/integer pairs.	L,n 1. L,2 1. <A[1,80] 2. <U2 or U4> . n. L,2 1. <A[1,80] 2. < U2 or U4>
SubstrateList	List of 0+ OBJID.	L, n 1. A[1,80] . n. A[1,80]
SubstIDRead	Text. Conforms to restrictions for SubstID in SEMI E90, SEMI E90.1.	A[1,80]
TransferState	Enumerated text: “ASSIGNED” “BLOCKED” “READY TO LOAD” “READY TO UNLOAD”	A[7,15]

7.4 *Data Table Object* — Table 9 specifies the attributes of the Data Table Object.

Table 9 Data Table Object Attribute Specification

Attribute Name	*Range/Value*	*SECS-II Format*
ObjType	Text = “TableData”	A[9]

Attribute Name	*Range/Value*	*SECS-II Format*
ObjID	Text. Conforms to restrictions for OBJID in SEMI E39.1, §6.	A[1,80]
RowTemplateID	OBJID for associated Row Template Object	A[1,80]

7.5 *IMM Data Table Object* — Table 10 specifies the attributes of the IMM Data Table Object. Note that the IMM Data Table Object inherits the attributes of a Data Table Object (Table 9).

Table 10 IMM Data Table Object Attribute Specification

Attribute Name	*Range/Value*	*SECS-II Format*
ObjType	Text = "TableIMM_Data".	A[13]
ObjID	Text. Conforms to restrictions for OBJID in SEMI E39.1, §6.	A[1,80]
DataValidity	Enumerated: 1 = Valid and Complete Data 2 = Incomplete Data 3 = Out-of-specification Data >3 reserved	U2
EventList	List of 0+ text/integer pairs.	L,n 1. L,2 1. <A[1,80] 2. <U2 or U4> . n. L,2 1. <A[1,80] 2. <U2 or U4>
IMM_ID	Text. Conforms to restrictions for OBJID in SEMI E39.1, §6.	A[1,80]
LotID	Text. Conforms to restrictions for OBJID in SEMI E39.1, §6.	A[1,80]
ProcessFlowStep	Text.	A[1,80]
ProcessRecipeID	Text. Conforms to restrictions for OBJID in SEMI E39.1, §6.	A[1,80]
ProcessToolID	Text. Conforms to restrictions for OBJID in SEMI E39.1, §6.	A[1,80]
RegionRanges	List of 2 integers.	L,2 1. <U2> 2. <U2>
Status	Enumerated: 1 = IN PROCESS 2 = IN RETENTION	U2
Substid	Text. Conforms to restrictions for SubstID in SEMI E90, SEMI E90.1.	A[1,80]
SubstidRead	Text. Conforms to restrictions for SubstID in SEMI E90, SEMI E90.1	A[1,80]
SubstrOrientation	Non-negative floating point.	F8
SubstrSide	Enumerated: 1 = Front 2 = Back 3 = Both	U2
Summary	List of 0+ SummaryData structures.	Ref. Table 11
TimeComplete	Timestamp format.	A[16]
TimeStart	Timestamp format.	A[16]
TimeStampFormat	Enumerated: 0 = Text > 0 Reserved	I1

7.5.1 Elements of the complex attributes of the IMM Data Table Object are mapped in Table 11.

Table 11 Elements of Complex Attributes of IMM Data Table Object

Attribute Name	*Range/Value*	*SECS-II Format*
RegionDef	List of 0+ structures consisting of RegionID, FirstRowID, FirstRowNum, LastRowNum.	L, n 1. L,4 1. <RegionID$_1$> 2. <FirstRowID$_1$> 3. <FirstRowNum$_1$> 4. <LastRowNum$_1$> . n. L,4 1. <RegionID$_n$> 2. <FirstRowID$_n$> 3. <FirstRowNum$_n$> 4. <LastRowNum$_n$>
RegionID	Text. For tables containing measurements from both sides of a substrate, the text shall begin with the string “Front” or the string “Back”.	A[1,80]
FirstRowID	Any integer or text.	A[1,80], In, Un
FirstRowNum	0–65,535	U2
LastRowNum	0–65,535	U2
SummaryData	Structure consisting of SummaryDataName, SummaryDataValue, SummaryDataForm, and SummaryDataUnits.	L, n 1. <SummaryDataName> 2. <SummaryDataValue> 3. <SummaryDataForm> 4. <SummaryDataUnits>
SummaryDataName	Text.	A[1,80]
SummaryDataValue	Any valid range. Dependent on IMM supplier documentation.	Any
SummaryDataForm	Enumerated per SEMI E5, §9.	Any
SummaryDataUnits	Enumerated text per SEMI E5, §9.	Alphanumeric text.

7.6 *IMM Calibration Data Table Object* — Table 12 specifies the attributes of the IMM Calibration Data Table Object.

Table 12 IMM Calibration Data Table Object Attribute Specification

Attribute Name	*Range/Value*	*SECS-II Format*
ObjType	Text = “TableIMM_CalibratedData”.	A[23]
ObjID	Text. Conforms to restrictions for OBJID in SEMI E39.1, §6.	A[1,80]
CalibrationDuration	0–65,535	U2
CalibrationStatus	Enumerated: 1 = Success 2 = Failed	U2
CalibrationTimeStamp	Timestamp format.	A[16]
RefSubstid	Text. Conforms to restrictions for SubstID in SEMI E90, SEMI E90.1	A[1,80]

7.7 *IMM Converted Data Table Object* — Note that the IMM Converted Data Table Object inherits the attributes of the IMM Data Table Object (Table 10). Table 13 specifies the additional attributes of the IMM Converted Data Table Object.

Table 13 IMM Converted Data Table Object Attribute Specification

Attribute Name	*Range/Value*	*SECS-II Format*
ObjType	Text = “TableIMM_ConvertedData”.	A[22]
ObjID	Text. Conforms to restrictions for OBJID in SEMI E39.1, §6.	A[1,80]
ConversionRecipeID	Text. Conforms to restrictions for OBJID in SEMI E39.1, §6.	A[1,80]
FeatureType	Enumerated text: “CD linewidth” “Thickness” “Reflectivity” “GoodnessOfFit” ….	A[]
RawDataTable	Text. Conforms to restrictions for OBJID in SEMI E39.1, §6.	A[1,80]

7.8 *IMM Raw Data Table Object* — Table 14 specifies the attributes of the IMM Raw Data Table Object. Note that the IMM Raw Data Table Object inherits the attributes of an IMM Data Table Object (Table 10)

Table 14 IMM Raw Data Table Object Attribute Specification

Attribute Name	*Range/Value*	*SECS-II Format*
ObjType	Text = “TableIMM_RawData”.	A[16]
ObjID	Text. Conforms to restrictions for OBJID in SEMI E39.1, §6.	A[1,80]
MeasurementRecipeID	Text. Conforms to restrictions for OBJID in SEMI E39.1, §6.	A[1,80]

7.9 *Data Row Object* — Table 15 specifies the attributes of the Data Row Object.

Table 15 Data Row Object Attribute Specification

Attribute Name	*Range/Value*	*SECS-II Format*
ObjType	Text = “DataRow”.	A[7]
ObjID	Text. Conforms to restrictions for OBJID in SEMI E39.1, §6.	A[1,80]
$DataItem_i$	Any except list of mixed types of data.	Array, A[], In, Un, Fn
RowNum	0–65,535	U2
RowOwner	A valid object specifier conforming to the form of ObjSpec in SEMI E39.	A[1,80]

7.10 *Data Row Template Object* — Table 16 specifies the attributes of the Data Row Template object.

Table 16 Data Row Template Object Attribute Specification

Attribute Name	*Range/Value*	*SECS-II Format*
ObjType	Text = “DataRow_Template”.	A[16]
ObjID	Text. Conforms to restrictions for OBJID in SEMI E39.1, §6.	A[1,80]
DataInfoIDList	List of 0+ text.	L, n 1. A[1,80] . 1. A[1,80]
NumEntries	0–65,535	U2

7.11 *Data Information Object* — Table 17 specifies the attributes of the Data Table object.

Table 17 Data Table Object Attribute Specification

Attribute Name	*Range/Value*	*SECS-II Format*
ObjType	Text = “DataInfo”.	A[8]
ObjID	Text. Conforms to restrictions for OBJID in SEMI E39.1, §6.	A[1,80]
Assurance	Any	
DataType	Enumerated text. arrays of integers =AR[In, Un] for n = 1, 2, 4, 8 arrays of floats = AR[F4, F8] ASCII = A[n] for n characters or A[min, max] for variable length strings Integers = I2, I4, I8, U2, U4, or U8 binary bytes = B[n] for n bytes Boolean = BOOLEAN	A[n]
Description	Text.	A[1,80]
InterdependencyRule	Text.	A[1,400]
Units	Enumerated per SEMI E5, §9.	A[1+]
Use	Text.	A[1,80]
ValidityRule	Text.	A[1,80]

7.12 *Additional Attributes of the Substrate Object* — Table 18 specifies the attributes of the Substrate object in addition to those specified in SEMI E90.

Table 18 Additional Substrate Object Attribute Specification

Attribute Name	*Range/Value*	*SECS-II Format*
ProcessFlowStep	Text.	A[1,80]
OrientationIn	Any	F8
OrientationOut	Any	F8
ProcessToolID	Text. Conforms to restrictions for OBJID in SEMI E39.1, §6.	A[1,80]
ProcessRecipeID	Text. Conforms to restrictions for OBJID in SEMI E39.1, §6.	A[1,80]
SubstrSide	Enumerated: Unknown = 0 Front = 1 Back = 2 Both = 3 >3 reserved	U2

8 Event Variables

8.1 Table 19 shows the mapping of event variables to SECS-II format. All event variables are DVVALs.

Table 19 Event Variables mapped to SECS-II

Event Variables	*Associated Object*	*Format*
DateTime	IMM, STPO	Timestamp format. A[16]
Service	IMM	U1
SubstrateListChangeType	IMM	U1 Enumerated: 1 = Addition 2 = Deletion
SubstrateID	IMM	A[1,80]. Conforms to restrictions for SubstID in SEMI E90, E90.1

SubstrLocStateChange	IMM	A[] Enumerated per SEMI E90.1
TableCount	IMM	U2
SubstrXfrPathID	STPO	A[1,80]
SubstrateList	STPO	L, n 1. <SubstrateID$_1$> . n. <SubstrateID$_n$>
SubstidRead	STPO	A[1,80]. Conforms to restrictions for SubstID in SEMI E90, E90.1
OrientationIn	STPO	F8
OrientationOut	STPO	F8

SEMI E128-0304
PROVISIONAL SPECIFICATION FOR XML MESSAGE STRUCTURES

This provisional specification was technically approved by the Global Information & Control Committee and is the direct responsibility of the North American Information & Control Committee. Current edition approved by the North American Regional Standards Committee on October 16, 2003 and December 4, 2003. Initially available at www.semi.org February 2004; to be published March 2004. Originally published July 2003.

1 Purpose

1.1 This standard specifies XML structures required to encode message header or "envelope" information for asynchronous messages. It applies to applications conforming to other SEMI standards only if they use asynchronous XML messaging for communication. XML is used in software applications to encode information with tags that provide structure and data type information specified in XML Schemas. XML is also used to structure information needed to route and deliver messages between applications. A standard for message headers is required to avoid development and use of multiple incompatible protocols based on XML. This specification will be referred to as "The XML Messaging Specification."

1.2 The XML Messaging Specification provides definitions of message headers needed for messages exchanged in an asynchronous fashion. Asynchronous messages are delivered independently without dependence on transport technology to correlate related messages. The rationale for choosing an asynchronous message delivery mechanism is based on the characteristics of the interaction between two software systems. Some of these reasons are that:

- Support for long-running Request/Reply interactions may not be practical in a synchronous session-oriented communication.
- Some messaging interactions require additional callback messages reporting on server activity which supplement the reply message.
- Clients may wish to interleave requests and replies to manage parallel or overlapping activities in multiple concurrent messaging conversations.

1.3 The base technology supporting asynchronous messaging may also support synchronous interactions that rely on transport sessions for identification and correlation of related messages. The XML Messaging Specification does not prevent these synchronous messaging dialogs since the specified header elements are not required in these cases. However, there may be benefits of using the message header elements specified here even in synchronous interactions. The message header elements provide self describing information about messages that may be useful for debugging or tracking message traffic. Their use may also enable future migration to different message transports.

1.4 The XML Messaging Specification uses established, openly referenceable industry standards for XML messaging where possible. It only specifies extensions to existing industry standards when needed to meet the immediate requirements for messaging in a SEMI Standard application context. The intent of this standard is not to replicate existing standards or offer competing specifications, but to align with and cite the usage of existing standards.

2 Scope

2.1 *In Scope* — The XML Messaging Specification will include **within its scope** the following:

2.1.1 *Use of the W3C Simple Object Access Protocol (SOAP)* — SOAP is used as a basic message foundation for SEMI messaging applications.

2.1.2 *Definition of SOAP Extensions* — This specification defines SOAP extensions for header and body elements needed to support asynchronous messaging interactions.

2.1.3 *Structures for Request/Reply Point-to-Point Message Dialogs* — This specification will address the message header data needed to support request/reply interactions with a request message and corresponding reply message using asynchronous, one-way messages.

2.1.4 *Structures for Asynchronous Point-to-Point Callback Messages* — This specification will address the message header data needed to enable atomic, asynchronous callback messages that complement a request/reply dialog between two endpoints.

2.1.5 *Structures for Asynchronous Multicast Event Messages* — This specification will address the message header data needed to enable atomic, asynchronous event messages distributed via a publication multicast to multiple subscribers.

2.1.6 *Use of W3C Web Services Definition Language (WSDL)* — WSDL is a message metadata specification language for describing signatures of web service interfaces. This specification will define how WSDL

will be used to represent the interfaces intended for asynchronous message exchange patterns.

2.2 *Out of Scope* — The following topics are **beyond the scope** of this specification. These topics may be addressed in separate specifications, but are not intended as future additions to this standard.

2.2.1 *Message Security* — The specification does not provide mechanisms for authentication of the message sender or receiver or for securing the content of messages from unauthorized access.

2.2.2 *Quality of Performance* — The XML Messaging Specification does not provide performance requirements or conformance testing for the performance characteristics of message transports. There may be environments in which XML messaging structures are not appropriate or effective for high performance communications. The definition of the environments where XML Messaging may or may not be appropriate is beyond the scope of this document.

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

3 Limitations

3.1 *Provisional Standard Deficiencies* — This document is Provisional due to areas where the specification needs additions. When these missing topics are addressed with subsequent revisions, this standard will be upgraded to remove Provisional status. The following items are cited as deficiencies in this document:

3.1.1 *SOAP 1.2* — This specification depends on the W3C SOAP standard. The release of the full SOAP 1.2 W3C Recommendation is anticipated in early 2003. Once it is formally adopted, this specification will need to make adjustments for changes between SOAP 1.1 and SOAP 1.2.

3.1.2 *Namespace for XML Messaging Schema* — This provisional specification will need to be aligned with a future guideline for the specification of namespaces belonging to SEMI Standards.

3.1.3 *Structure for Publish/Subscribe Event Messages* — This provisional specification does not yet provide mechanisms for defining messages for distribution via a multicast publish/subscribe mechanism.

3.1.4 *Structure for Attachments with Non-XML Data* — This provisional specification does not yet specify the message structure for messages that have both a SOAP envelope and a message attachment containing non-XML data.

3.1.5 *WSDL Requirements* — The provisional specification does not yet specify the form of WSDL required for exposing services for access using asynchronous messaging. Additional investigation into WSDL and SOAP 1.2 Message Exchange Patterns is required before completing the WSDL section of this specification.

4 Referenced Standards

4.1 *World Wide Web Consortium (W3C) Standards*[1]

Extensible Markup Language (XML) 1.0 (Second Edition) — W3C Recommendation, 6 October 2000. (See http://www.w3.org/TR/2000/REC-xml-20001006.) The latest version of Extensible Markup Lanuage (XML) 1.0 is available at http://www.w3.org/TR/REC-xml.

Namespaces in XML — W3C Recommendation, 14 January 1999. (See http://www.w3.org/TR/1999/REC-xml-names-19990114/.) The latest version of Namespaces in XML is available at http://www.w3.org/TR/REC-xml-names.

XML Schema Part 1: Structures — W3C Recommendation, 2 May 2001. (See http://www.w3.org/TR/2001/REC-xmlschema-1-20010502/.) The latest version of XML Schema Part 1: Structures is available at http://www.w3.org/TR/xmlschema-1/

XML Schema Part 2: Datatypes — W3C Recommendation, 2 May 2001. (See http://www.w3.org/TR/2001/REC-xmlschema-2-20010502/.) The latest version of XML Schema Part 2: Datatypes is available at http://www.w3.org/TR/xmlschema-2/.

Simple Object Access Protocol 1.1 (SOAP) — W3C Note, 8 May 2000. See http://www.w3.org/TR/2000/NOTE-SOAP-20000508). The latest version of Simple Object Access Protocol 1.1 is available at http://www.w3.org/TR/SOAP.

Web Services Description Language 1.1 (WSDL) — World Wide Web Consortium, 15 March 2002. This version of the Web Services Description Language Note is http://www.w3.org/TR/2001/NOTE-wsdl-20010315. The latest version of Web Services Description Language is available at http://www.w3.org/TR/wsdl.

1 All standards available from the website: www.w3.org. World Wide Web Consortium, Massachusetts Institute of Technology (MIT) Laboratory for Computer Science, 200 Technology Square, Cambridge, MA 02139, USA. Telephone: 1.617.253.2613 Fax: 1.617.258.5999

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

5 Terminology

5.1 *Abbreviations and Acronyms*

5.1.1 *SOAP* — Simple Object Access Protocol

5.1.2 *WSDL* — Web Services Definition Language

5.1.3 *XML* — Extensible Markup Language

5.2 *Definitions*

5.2.1 *asynchronous messaging* — a style of communication based on the exchange of atomic messages separated in time and implemented with one-way message deliveries.

5.2.2 *callback message* — a message that communicates supplemental information resulting from performance of an action initiated by a related request/reply conversation.

5.2.3 *message document* — an XML document that contains the message envelope and encapsulated message header and message content.

5.2.4 *message envelope* — the encapsulating XML structures that define an overall framework for expressing what is in a message; who should deal with it, and whether it is optional or mandatory.

5.2.5 *reply message* — a message that contains data resulting from the completion of an action initiated by a related request message.

5.2.6 *request message* — a message that contains necessary information to allow a server to perform a requested action on behalf of the requester.

6 Requirements

6.1 *Simple Object Access Protocol (SOAP) Envelope* — Messages shall be enclosed in the Envelope element as specified in the SOAP 1.1 document. The SOAP Envelope shall include a Header element and a Body element. Within the Header element, message documents shall include an element named MessageHeader that contains data required for message preparation and transport as specified below.

6.2 *SEMI XML Messaging Namespace* — All XML items defined in this specification belong to the reserved SEMI namespace "urn:semi-org:schema:xmlmsg:0:0". The following XML Namespace declaration shall be used to specify the namespace.

```
<xsd:schema xmlns="urn:semi-
org:schema:xmlmsg:0:0"
elementFormDefault="qualified" >
```

6.3 *Use of SOAP Messages for Synchronous Request/Reply Interactions* — The XML Messaging Specification does not preclude the use of SOAP for synchronous interactions. The SOAP Header Extensions specified in this document are not required for a synchronous exchange, but may be useful even in those cases. A standard adopting a synchronous use of SOAP shall clearly indicate whether the header extensions are required for conformant implementations.

6.4 *SOAP Header Entry for Messaging* — The SOAP message shall have an entry contained in the SOAP-ENV:Header element named **MessageHeader** that contains elements as defined in this section. The MessageHeader element shall be defined with a mustUnderstand = "1" attribute that requires all conformant implementations to process the header element or return a "MustUnderstand" fault to the message sender.

6.4.1 *From* — The **From** element shall identify the application that sent this message. The format of the value in the **From** element shall be a valid URI. The domain of the value in the **From** element may vary by transport binding.

6.4.2 *To* — The **To** element shall identify the application that is the intended recipient of this message. The format of the value in the **To** element shall be a valid URI. The domain of the value in the **To** element may vary by transport binding.

6.4.3 *MessageType* — The **MessageType** element shall identify the role of the message in a conversation. A message shall be either a request, reply or callback message.

6.4.4 *Request Message* — A **request** message shall originate from an endpoint in the client role and shall establish a new RequestId. The RequestId shall be unique within the domain of all such Ids generated by the originating client. The request message body contains the information needed by the server to enable it to perform a requested action on behalf of the client.

6.4.4.1 *Reply Message* — A **reply** message shall originate from an endpoint in the server role and shall echo the RequestId received from a client in a prior request message as a CorrelationId. The reply message body may contain information that conveys the completion of an action initiated by a related request message. It may instead include a body entry for a Fault that communicates the reason for a failure. In other cases the Fault may provide information such as

warnings that supplement the normal content of the body.

6.4.4.2 *Callback Message* — A **callback** message shall originate from an endpoint in the server role and shall echo the RequestId received from a client in a prior request message. A callback message body contains unspecified information that relates to the requested action initiated by a related request message. A callback message may precede the reply message (a leading callback message), or it may follow the reply message (a trailing callback message). The callback message shall also contain an element for the EventIndex if the server will send more than one callback message. The body of a callback message may also contain SOAP Faults if there is a processing error that needs to be conveyed to the client.

6.4.5 *RequestId* — The **RequestId** element shall be used in a request message to identify it as the first of a series of messages that belong to a single interaction. The value of RequestId shall be generated by a client with the initial request message and then shall be echoed back as the **CorrelationId** in subsequent callback messages or reply messages from the responding server. The XML Schema shall specify a choice between RequestId and CorrelationId. RequestId shall be the choice when MessageType is "Request".

6.4.6 *CorrelationId* — The **CorrelationId** element shall be used in each reply or callback message to identify it as part of an interaction that was initiated by a prior request message. The value of CorrelationId shall be identical to the **RequestId** from the initial request message. The same CorrelationId value shall be echoed back in all callback messages or reply messages sent from the server in response to the client's initial request message. The XML Schema shall specify a choice between RequestId and CorrelationId. CorrelationId shall be chosen when MessageType is "Reply" or "Callback".

6.4.7 *Action* — The **Action** element shall be a string that serves to identify the unit of functionality invoked by the request message. An Action value shall be unique to the recipient of the message.

6.4.8 *ReplyExpected* — The **ReplyExpected** element is optional. When present, the ReplyExpected element shall indicate whether the client expects to receive a reply message by a "true" value. When present with a value of "false", the ReplyExpected element shall indicate that the client does not expect to receive a reply message. If not present, the server receiving the request shall assume that a reply is expected (equivalent to a "true" value). A service shall be prepared to return a reply for every request, and a client shall be prepared to receive replies even if this element is set to "false". ReplyExpected is intended only for optimizing conversations where the client intends to ignore reply messages.

6.4.9 *MessageId* — The **MessageId** element shall be optional in the MessageHeader to define a string value to uniquely identify the message. MessageId need not be included in the MessageHeader if there is no value established for uniquely identifying each message by the selected message transport. If present, MessageId shall be globally unique.

6.4.10 *EventIndex* — The **EventIndex** element shall be optional in the MessageHeader of a callback message. EventIndex shall not be included in request or reply messages. Within the MessageHeader of a callback message, EventIndex is optional only if there is just one callback message in an interaction, but required if there are a series of callbacks. EventIndex is used to identify this message's position in relation to multiple callback messages resulting from the original request.

6.4.10.1 *EventIndex Sub-elements* — EventIndex shall contain sub-elements for Position, and either Total or Finished as specified in the following paragraphs. Either Total or Finished shall be present, but not both.

6.4.10.2 *Position* — The **Position** element shall indicate this message's position in a stream of callback messages. Position shall be greater than or equal to 1. Position shall be less than or equal to Total, when Total is present.

6.4.10.3 *Total* — The **Total** element shall indicate the final count of callback messages to be delivered in this interaction (that is, all messages sharing a common RequestId/CorrelationId). Total shall have the same value in each callback message in the sequence.

6.4.10.4 *Finished* — The **Finished** element shall indicate whether this is the last in a sequence of callback messages. Finished shall be present in callback messages for interactions in which the total number of callback messages cannot be known in advance. A value of "false" shall indicate that additional callback messages will be sent. A value of "true" shall indicate that no further callback messages will be sent.

6.4.10.5 *Default Interpretation if EventIndex is not Present* — If EventIndex is not present, the client receiving the callback message shall interpret it as equivalent to a EventIndex element with Position = 1 and Total = 1, indicating a single callback message from server to client.

6.4.11 *MessageHeader XML Schema* — The XML schema in Figure 1 defines the MessageHeader element that is required by this specification.

```
<?xml version="1.0" encoding="UTF-8"?>
<xsd:schema targetNamespace="urn:semi-org:schema:xmlmsg:0:0"
xmlns:xsd="http://www.w3.org/2001/XMLSchema" elementFormDefault="qualified">
   <xsd:element name="MessageHeader">
      <xsd:complexType>
         <xsd:sequence>
            <xsd:element name="From" type="xsd:anyURI"/>
            <xsd:element name="To" type="xsd:anyURI"/>
            <xsd:element name="MessageType">
               <xsd:simpleType>
                  <xsd:restriction base="xsd:string">
                     <xsd:enumeration value="REQUEST"/>
                     <xsd:enumeration value="REPLY"/>
                     <xsd:enumeration value="CALLBACK"/>
                  </xsd:restriction>
               </xsd:simpleType>
            </xsd:element>
            <xsd:choice>
               <xsd:element name="RequestId" type="xsd:string"/>
               <xsd:element name="CorrelationId" type="xsd:string"/>
            </xsd:choice>
            <xsd:element name="Action" type="xsd:string"/>
            <xsd:element name="ReplyExpected" type="xsd:boolean" minOccurs="0"/>
            <xsd:element name="MessageId" type="xsd:string" minOccurs="0"/>
            <xsd:element name="EventIndex" minOccurs="0">
               <xsd:complexType>
                  <xsd:sequence>
                     <xsd:element name="Position" type="xsd:nonNegativeInteger"/>
                     <xsd:choice>
                        <xsd:element name="Total" type="xsd:nonNegativeInteger"/>
                        <xsd:element name="Finished" type="xsd:boolean"/>
                     </xsd:choice>
                  </xsd:sequence>
               </xsd:complexType>
            </xsd:element>
         </xsd:sequence>
      </xsd:complexType>
   </xsd:element>
</xsd:schema>
```

Figure 1
Message Header XML Schema

6.5 *SOAP Body Extensions* — Any exceptions that occur shall be communicated with the use of SOAP Faults.

6.5.1 *SOAP Faults* — A SOAP **Fault** element shall be inserted into the SOAP body element as a body entry to relay error status from the server back to the client. A body entry is a top level element within the Body element of the SOAP envelope. SOAP Fault subelements are specified in SOAP 1.1 Section 4.4. Their usage is specified here.

6.5.1.1 *fault* — The fault element shall appear in the body of the SOAP envelope as a body entry whenever there is an abnormal outcome in the server's processing of a request message.

6.5.1.2 *faultcode* — The faultcode element shall be present in a SOAP fault element. The values specified in the SOAP specification shall be used to indicate the nature of the error that occurred.

- "Client" indicates that the request message was not correct and could not be processed by the server.
- "Server" indicates that the request message was valid, but the server could not complete the requested action.

6.5.1.3 *faultstring* — The faultstring element shall be present to describe the cause of the fault in human understandable text.

6.5.1.4 *faultactor* — The use of the faultactor element is optional.

6.5.1.5 *detail* — The detail element is optional. It may be used to describe the cause of a fault that resulted from processing the body of the request message.

6.6 *XML Message Examples* — The examples of XML messages presented in this section are intended to help the reader understand the structure of messages that conform to this specification.

6.6.1 *Example SOAP Envelope* — The SOAP envelope illustrated in Figure 2 contains the required Header and Body elements. It also includes the MessageHeader element specified in this standard. The contents of these elements are intentionally left empty in this example.

```
<?xml version="1.0" encoding="UTF-8"?>
<SOAP-ENV:Envelope xmlns:SOAP-
ENV="http://schemas.xmlsoap.org/soap/envelope/">
  <SOAP-ENV:Header>
    <MessageHeader>
    </MessageHeader>
  </SOAP-ENV:Header>
  <SOAP-ENV:Body>
    <PayloadData>
    </PayloadData>
  </SOAP-ENV:Body>
</SOAP-ENV:Envelope>
```

Figure 2
SOAP Envelope

6.6.2 *Example Request Message* — The message example shown in Figure 3 illustrates the completed MessageHeader for a message from a client to a server to request a service. In this example, the SOAP Body contains an element that identifies the data that is being requested with the Action “dataRequest.” The RequestId assigned by the client is “7.” The client will expect a reply message that contains “7” in the CorrelationId.

```
<?xml version="1.0" encoding="UTF-8"?>
<SOAP-ENV:Envelope xmlns:SOAP-
ENV="http://schemas.xmlsoap.org/soap/envelope/">
  <SOAP-ENV:Header>
    <MessageHeader xmlns="urn:semi-org:schema:xmlmsg:0:0"
        elementFormDefault="qualified">
      <From>EqHost</From>
      <To>EQ99</To>
      <MessageType>REQUEST</MessageType>
      <RequestId>7</RequestId>
      <Action>dataRequest</Action>
      <ReplyExpected>true</ReplyExpected>
    </MessageHeader>
  </SOAP-ENV:Header>
  <SOAP-ENV:Body>
    <data>
      <DATAID type="ASC">420</DATAID>
    </data>
  </SOAP-ENV:Body>
</SOAP-ENV:Envelope>
```

Figure 3
Request Message Example

6.6.3 *Example Successful Reply Message* — The message example shown in Figure 4 illustrates the completed MessageHeader for a reply message from a server to a client with a response to a prior request message. This example represents a reply to the request message in the previous example. The CorrelationId of "7" serves to associate this reply with the RequestId of the earlier request message.

```
<?xml version="1.0" encoding="UTF-8"?>
<SOAP-ENV:Envelope xmlns:SOAP-
ENV="http://schemas.xmlsoap.org/soap/envelope/">
  <SOAP-ENV:Header>
    <MessageHeader xmlns="urn:semi-org:schema:xmlmsg:0:0"
        elementFormDefault="qualified">
      <From>EQ99</From>
      <To>EqHost</To>
      <MessageType>REPLY</MessageType>
      <CorrelationId>7</CorrelationId>
      <Action>dataRequest</Action>
    </MessageHeader>
  </SOAP-ENV:Header>
  <SOAP-ENV:Body>
    <data>
      <DATAID type="ASC">420</DATAID>
      <CEID type="ASC">GAS</CEID>
      <DATASETS type="LIST">
        <DATASET>
        </DATASET>
      </DATASETS>
    </data>
  </SOAP-ENV:Body>
</SOAP-ENV:Envelope>
```

Figure 4
Reply Message (Successful) Example

6.6.4 *Example Reply Message for a Bad Request* — In Figure 5, a faultcode value of “SOAP-ENV:Client” indicates that the error was caused by bad format or data in the request message from the client.

```
<?xml version="1.0" encoding="UTF-8"?>
<SOAP-ENV:Envelope xmlns:SOAP-
ENV="http://schemas.xmlsoap.org/soap/envelope/">
  <SOAP-ENV:Header>
    <MessageHeader xmlns="urn:semi-org:schema:xmlmsg:0:0"
        elementFormDefault="qualified">
      <From>EQ99</From>
      <To>EqHost</To>
      <MessageType>REPLY</MessageType>
      <CorrelationId>7</CorrelationId>
      <Action>dataRequest</Action>
    </MessageHeader>
  </SOAP-ENV:Header>
  <SOAP-ENV:Body>
    <SOAP-ENV:Fault>
      <faultcode>SOAP-ENV:Client</faultcode>
      <faultstring>BadSyntax</faultstring>
      <detail>DATAID not found in request.</detail>
    </SOAP-ENV:Fault>
</SOAP-ENV:Envelope>
```

Figure 5
Reply Message (Bad Request) Example

6.6.5 *Example Reply Message for a Processing Error* — In Figure 6, a faultcode value of “SOAP-ENV:Server” indicates that the error occurred during processing of the message. The request message was complete and valid to allow the server to begin processing, but some occurrence prevented successful completion of the action requested by the client.

```
<?xml version="1.0" encoding="UTF-8"?>
<SOAP-ENV:Envelope xmlns:SOAP-
ENV="http://schemas.xmlsoap.org/soap/envelope/">
  <SOAP-ENV:Header>
    <MessageHeader xmlns="urn:semi-org:schema:xmlmsg:0:0"
      elementFormDefault="qualified">
      <From>EQ99</From>
      <To>EqHost</To>
      <MessageType>REPLY</MessageType>
      <CorrelationId>7</CorrelationId>
      <Action>dataRequest</Action>
    </MessageHeader>
  </SOAP-ENV:Header>
  <SOAP-ENV:Body>
    <SOAP-ENV:Fault>
      <faultcode>SOAP-ENV:Server</faultcode>
      <faultstring>IdNotFound</faultstring>
      <detail>DATAID requested not known to tool.</detail>
    </SOAP-ENV:Fault></SOAP-ENV:Body>
  </SOAP-ENV:Body>
</SOAP-ENV:Envelope>
```

Figure 6
Reply Message (Processing Error) Example

6.6.6 *Example Callback Message for a Variable Number of Events* — Figure 7 shows a callback message from a server to a client. The callback delivers some additional data related to a prior request message. The CorrelationId "8" allows the client to correlate the callback with a prior request message. This callback uses the EventIndex element to convey that this callback is the third of a variable number of callbacks. When the value of Finished is "true" then the client will know not to expect any further callback messages with this CorrelationId.

```
<?xml version="1.0" encoding="UTF-8"?>
<SOAP-ENV:Envelope xmlns:SOAP-
ENV="http://schemas.xmlsoap.org/soap/envelope/">
  <SOAP-ENV:Header>
    <MessageHeader xmlns="urn:semi-org:schema:xmlmsg:0:0"
    elementFormDefault="qualified">
      <From>EQ99</From>
      <To>EqHost</To>
      <MessageType>CALLBACK</MessageType>
      <CorrelationId>8</CorrelationId>
      <Action>dataSubscribe</Action>
      <EventIndex>
        <Position>3</Position>
        <Finished>false</Finished>
      </EventIndex>
    </MessageHeader>
  </SOAP-ENV:Header>
  <SOAP-ENV:Body>
    <data>
      <DATAID type="ASC">420</DATAID>
      <CEID type="ASC">GAS</CEID>
      <DATASETS type="LIST">
        <DATASET>
          <DSID>A</DSID>
          <DVS>
            <DV>
              <NAME>ARGON</NAME>
              <VAL>156</VAL>
            </DV>
          </DVS>
        </DATASET>
        <DATASET>
          <DSID>B</DSID>
          <DVS>
            <DV>
              <NAME>ARGON</NAME>
              <VAL>67</VAL>
            </DV>
          </DVS>
        </DATASET>
      </DATASETS>
    </data>
  </SOAP-ENV:Body>
</SOAP-ENV:Envelope>
```

Figure 7
Callback Message (Variable Number of Events) Example

6.6.7 *Example Callback Message for a Fixed Number of Events* — Figure 8 shows another callback message from a server to a client. The CorrelationId "9" allows the client to correlate the callback with a prior request message. The EventIndex element establishes this callback as the third of five callbacks. When the value of Position equals the value of Total, then the client will know not to expect any further callback messages with this CorrelationId.

```
<?xml version="1.0" encoding="UTF-8"?>
<SOAP-ENV:Envelope xmlns:SOAP-
ENV="http://schemas.xmlsoap.org/soap/envelope/">
  <SOAP-ENV:Header>
    <MessageHeader xmlns="urn:semi-org:schema:xmlmsg:0:0"
      elementFormDefault="qualified">
      <From>EQ99</From>
      <To>EqHost</To>
      <MessageType>CALLBACK</MessageType>
      <CorrelationId>9</CorrelationId>
      <Action>dataSubscribe</Action>
      <EventIndex>
        <Position>3</Position>
        <Total>5</Total>
      </EventIndex>
    </MessageHeader>
  </SOAP-ENV:Header>
  <SOAP-ENV:Body>
    <data>
      <DATAID type="ASC">420</DATAID>
      <CEID type="ASC">GAS</CEID>
      <DATASETS type="LIST">
        <DATASET>
        </DATASET>
      </DATASETS>
    </data>
  </SOAP-ENV:Body>
</SOAP-ENV:Envelope>
```

Figure 8
Callback Message (Fixed Number of Events) Example

7 Related Documents

None.

RELATED INFORMATION 1
USAGE SCENARIOS

NOTICE: This related information is not an official part of SEMI E128 and was derived from the work of the originating committee. This related information was approved for publication by full letter ballot procedures.

R1-1 Request/Reply (Synchronous)

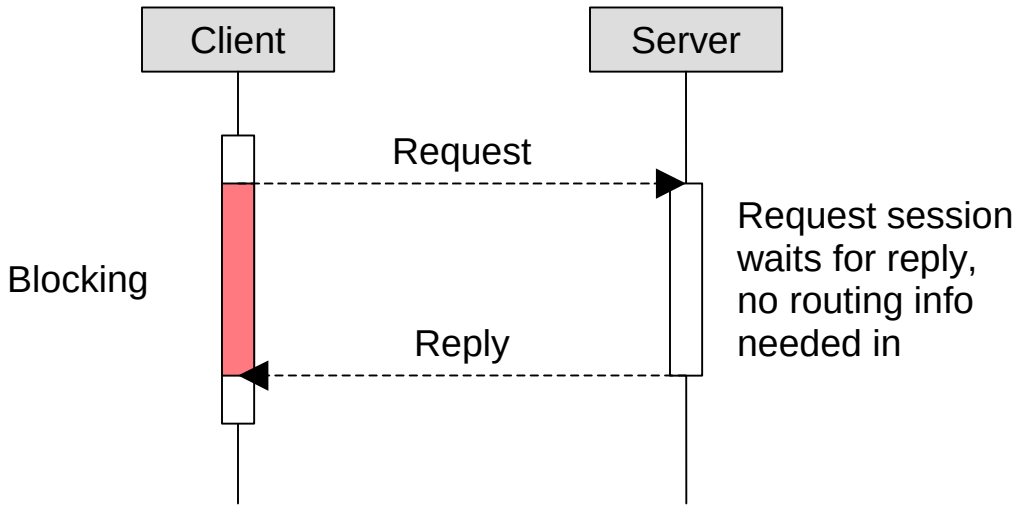

Figure R1-1
Synchronous Request/Reply Interaction

R1-1.1 The synchronous form of request reply shows the client blocking after sending a request message to wait for the reply message. The transport used in a synchronous interaction typically manages the correlation of the request with the reply by completing the interaction in a single step. Synchronous interactions have the advantage of simplicity.

R1-1.2 Synchronous interactions often also come with limitations. The client may not perform other activities while waiting for the reply. If the requested action takes a long time, this can limit the client's ability to respond to other events. Another limitation is the risk of having a failure occur in the server that is not reported to the client. In general, synchronous interactions may prove more fragile than asynchronous messaging in distributed systems.

R1-2 Request/Reply (Asynchronous)

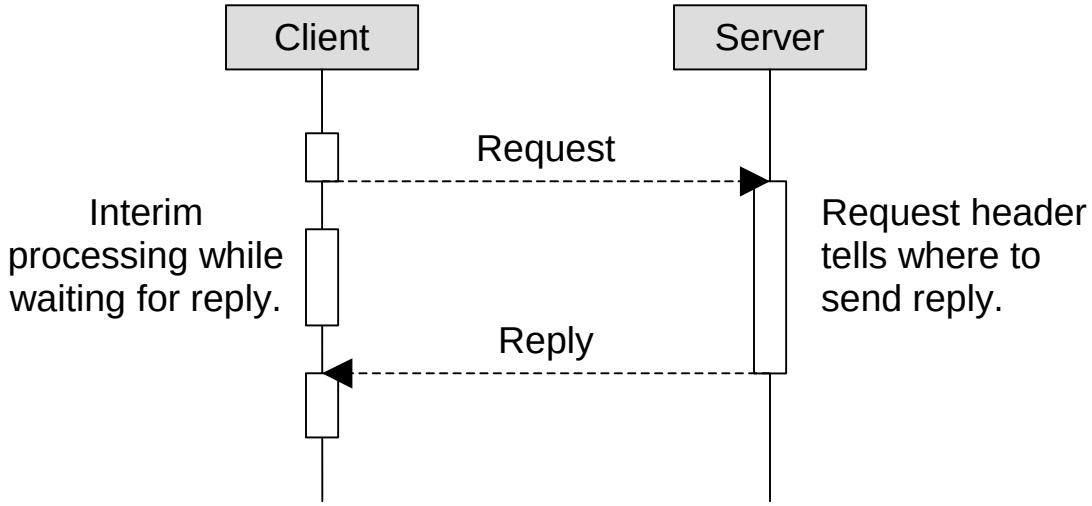

Figure R1-2
Asynchronous Request/Reply Interaction

R1-2.1 An asynchronous interaction can also support the semantics of a request/reply interaction. The request message includes envelope header data that enables the service to perform the requested action and then construct a "callback" reply message that is delivered to the request originator in a separate message delivery. In addition to data identifying the request and reply endpoints, the message envelope header must include a request identity that allows the requestor to correlate the reply message with the previously sent request message.

R1-3 Request/Reply with Interleaved Requests

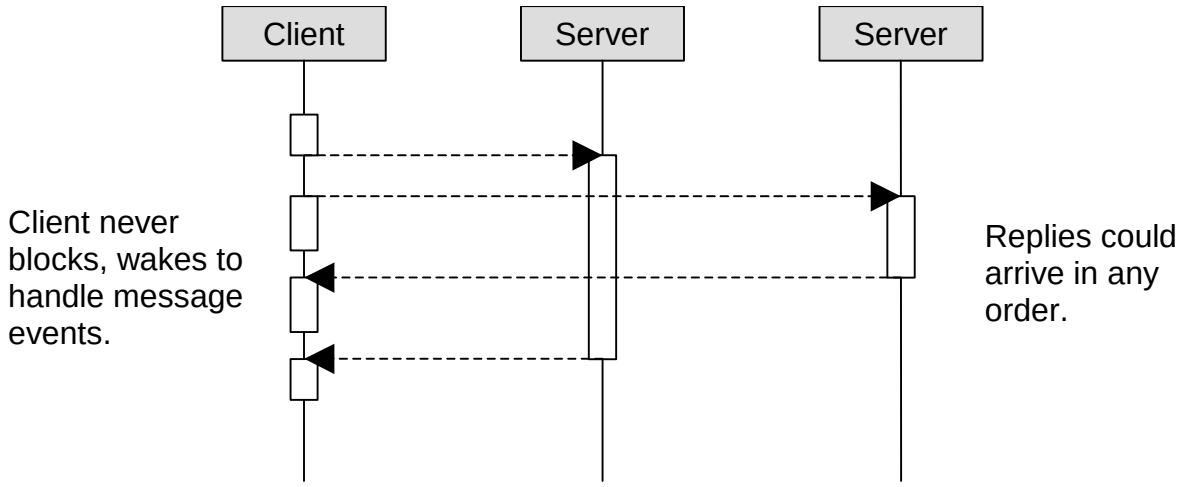

Figure R1-3
Interleaved Requests Interaction

R1-3.1 An asynchronous messaging interaction may involve more than one request being processed at a time. In this example, the Client requests actions from two separate Servers. The servers perform the actions and reply with the results independently. The client receives the reply message in a non-deterministic order. In some cases, the client may need to wait for both responses to proceed with its processing. This mechanism for parallel requests is a

natural use of asynchronous messaging that does not require the complexities of multi-threaded implementations and thread-safe programming.

R1-4 Request/Reply with Trailing Callbacks

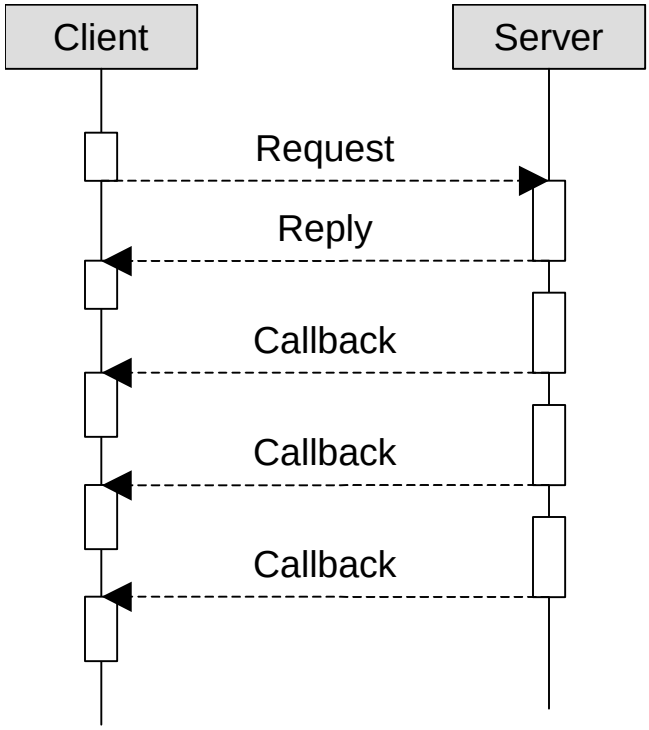

Figure R1-4
Request with Trailing Callback Interaction

R1-4.1 A request may ask a Server to establish some activity that generates a series of callback messages. The Client may just require a reply that acknowledges that the activity has been initiated with a trailing sequence of independent messages that deliver the data of interest to the Client. This message pattern uses the RequestId from the request message and the matching CorrelationId in the reply message to acknowledge the contract to deliver data in the callback messages that follow. The use of a message body element for the MessageSequence enables the Server to dynamically communicate the message sequence and establish intent to generate additional callback messages.

R1-5 Request/Reply with Leading Callbacks

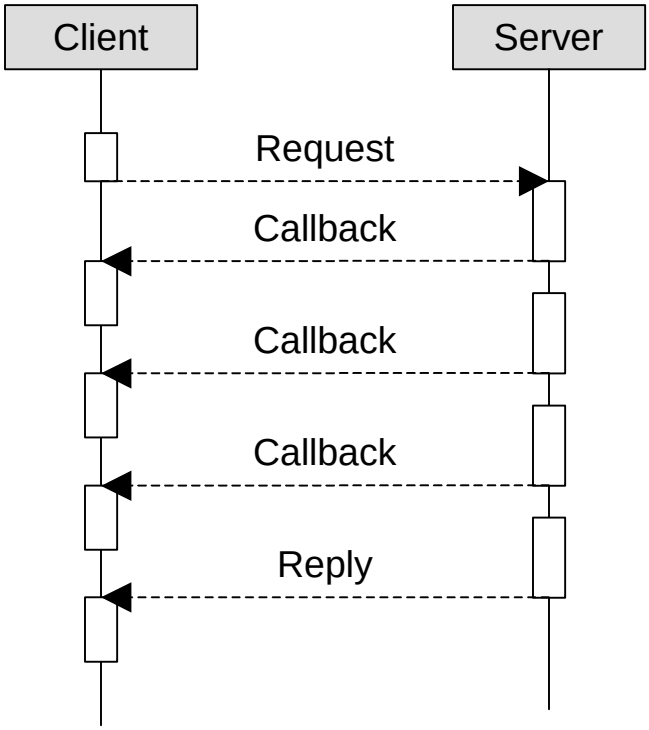

Figure R1-5
Request with Leading Callback Interaction

R1-5.1 Another common message exchange pattern involves a request message, followed by a series of callback messages providing status updates, with a final reply message that ends the interaction. This pattern is often used in an application where there are events that might be of interest that occur during the performance of the requested action. It is particularly useful if the Client may need to make decisions or take other actions based on the interim callback messages that are provided in lead of the reply.

R1-6 Request/Reply with Timeout

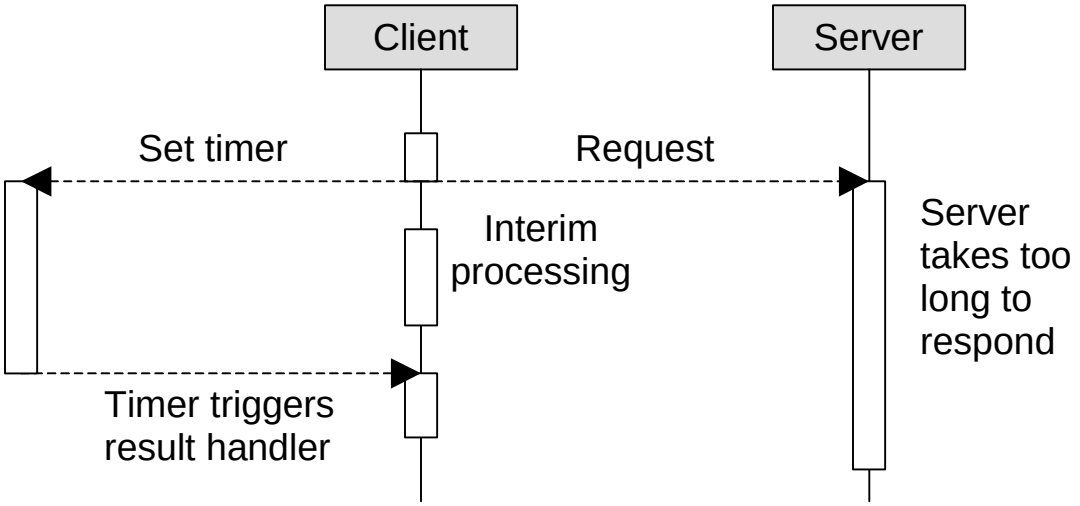

Figure R1-6
Request/Reply with Timeout Interaction

R1-6.1 A common messaging pattern that takes advantage is a request/reply that enables the Client to take alternate action if a reply is not received within a predefined time interval. This enables fault tolerant design that ensures that the client can recover from errors that prevent timely, or any, response from the server. The mechanism for the triggering of the timeout can itself be based on a messaging interaction with the request setting a timer and the reply reporting the timer expiration.

RELATED INFORMATION 2
MESSAGE TRANSPORT EXAMPLES

NOTICE: This related information is not an official part of SEMI E128 and was derived from the work of the originating committee. This related information was approved for publication by full letter ballot procedures.

R2-1 Transport Independence

R2-1.1 The aim of these examples is to keep implementation and transport mechanism specifics separated from the use of the XML Messaging Specification. The example interface consists of five operations:

- start(myName) — starts the messaging send/receive endpoint,
- send(destName, envelope) — sends a message to another endpoint,
- stop()— terminates the messaging endpoint,
- onMessage(envelope) — callback provided by an application to process incoming messages, and
- receive() — waits for next message (needed when onMessage is not supported).

R2-1.2 This kind of interface abstracts the minimum functionality needed to send and receive messages. Any potential message transport (HTTP, JMS, HSMS, RMI, etc.) could be encapsulated into methods similar to these to keep the messaging implementation independent of the transport. An application could then be written using HTTP today and then easily switched to JMS a year later. Also, new applications could be prototyped with lightweight transports and deployed with robust MOM transports with no change to application code.

R2-1.3 This Related Information is based on a simple example (illustrated in Figure R2-1) of a typical interaction between a client application sending requests to a server which provides the service and sends reply messages. The purpose of this example is to illustrate the way the message header elements proposed in this specification are realized in an asynchronous messaging implementation over HTTP.

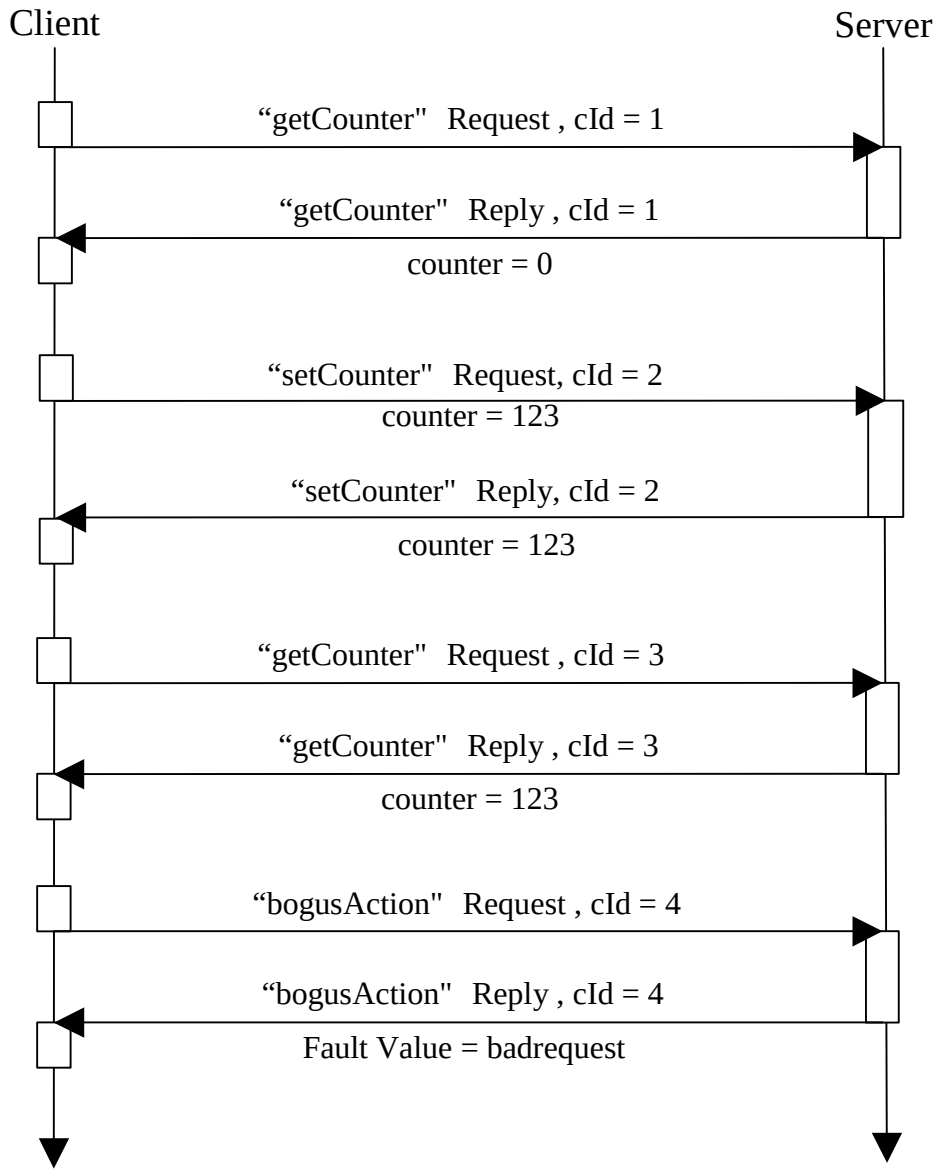

Figure R2-1
Sample Interaction for Messaging Examples

R2-1.4 In order to show this example in an application context, it was implemented in Java for both HTTP and JMS transports. The examples that follow were extracted from this sample application and the debug output of the running example.

R2-2 SOAP/HTTP

R2-2.1 Since HTTP is inherently synchronous, a common implementation would support only synchronous request/reply and would not use any of the MessageHeader elements that support asynchronous messaging. In the synchronous case, only the **To** and **Action** elements would be present in the MessageHeader. The **To** element would map to the URL of the target service and **Action** could also be URL encoded.

R2-2.2 To implement asynchronous messages over HTTP, both client and server endpoints must listen for messages coming through the selected transport. Each message is interpreted based on both the transport properties and the XML MessageHeader included in the SOAP header. Unlike more typical HTTP usage, the request and reply are not

constrained to a single session context, and can be separated in time. There is no “blocking” on pending replies. This model lends itself better to clients or servers that deal with many messaging partners at a time, while still needing to act on common data.

R2-2.3 In the HTTP/SOAP transport example, each of the messages in the sample scenario will be sent with an HTTP Post operation. Every message will include an XML document containing a SOAP envelope and the MessageHeader specified in this standard. The message body will contain the application specific elements that communicate the content of each message.

R2-2.4 The elements of the MessageHeader contain data that is used to construct the appropriate HTTP message and to populate the properties of the SOAP message. Specifically, the destination URL for each message appears in the To attribute of the MessageHeader. The From attribute contains the URL of the sender of each message. The SOAPAction HTTP header replicates the Action element in the transport specific header attribute.

R2-2.5 This example (and the ones that follow) include both the HTTP properties and the XML message from the client perspective. The request message shown in Figure R2-2 illustrates the HTTP transport form of the message.

```
Sending HTTP POST:
POST getCounter HTTP/1.1
Host: localhost:9001
Content-Type: application/soap+xml; charset="utf-8"
Content-Length: 369
SOAPAction: getCounter

<Envelope>
    <Header>
        <MessageHeader>
            <From>localhost:9001</From>
            <To>localhost:9000</To>
            <MessageType>REQUEST</MessageType>
            <RequestId>1</RequestId>
            <Action>getCounter</Action>
        </MessageHeader>
    </Header>
    <Body />
</Envelope>
```

Figure R2-2
Example HTTP Post Request Message

R2-2.6 The reply message in Figure R2-3 follows the same pattern, with the TO header element reflected in the HTTP Post and the SOAPAction mapping to the MessageHeader Action element. In this example, the same Action value is echoed back with the reply message.

```
Received HTTP POST:
POST getCounter HTTP/1.1
Host: localhost:9000
Content-Type: application/soap+xml; charset="utf-8"
Content-Length: 416
SOAPAction: getCounter

<Envelope>
    <Header>
        <MessageHeader>
            <From>localhost:9000</From>
            <To>localhost:9001</To>
            <MessageType>REPLY</MessageType>
            <CorrelationId>1</CorrelationId>
            <Action>getCounter</Action>
        </MessageHeader>
    </Header>
    <Body>
        <counter>0</counter>
    </Body>
</Envelope>
```

Figure R2-3
Example HTTP Post Reply Message

R2-3 Java Message Service

R2-3.1 The Java Message Service example is based on the same example application, but with a JMS implementation. This example shows how message-oriented middleware can be used to provide persistent and guaranteed messaging, without changing application code. All transport-specific messaging is confined to the Java classes that implement the example. Application code is only exposed to the XML messaging transport interface and the XML Envelope specification.

R2-3.2 The JMS header properties illustrated below show how the XML messaging elements are mapped to JMS. Figure R2-4 shows the headers for the request and the reply message as they map to the JMS header properties.

```
==========================================
Sending JMS Message:
JMS Header Property: To=myServer
JMS Header Property: Action=getCounter
JMS Header Property: From=myClient
JMS Header Property: RequestId=1
JMS Header Property: MessageType=REQUEST
JMS Message Body: <Body />
COMMAND:
==========================================
Receiving JMS Message:
JMS Header Property: To=myClient
JMS Header Property: CorrelationId=1
JMS Header Property: Action=getCounter
JMS Header Property: From=myServer
JMS Header Property: MessageType=REPLY
JMS Message Body: <Body>
    <counter>0</counter>
</Body>
==========================================
Message received:
<Envelope>
    <Header>
        <MessageHeader>
            <To>myClient</To>
            <CorrelationId>1</CorrelationId>
            <Action>getCounter</Action>
            <From>myServer</From>
            <MessageType>REPLY</MessageType>
        </MessageHeader>
    </Header>
    <Body>
        <counter>0</counter>
    </Body>
</Envelope>
```

Figure R2-4
Example JMS Message

SEMI E130-1104
SPECIFICATION FOR PROBER SPECIFIC EQUIPMENT MODEL FOR 300 mm ENVIRONMENT (PSEM300)

This specification was technically approved by the Global Information & Control Committee and is the direct responsibility of the North American Information & Control Committee. Current edition approved by the North American Regional Standards Committee on September 3, 2003. Initially available at www.semi.org October 2003; to be published November 2003.

NOTICE: The designation of SEMI E130 was updated during the 1104 publishing cycle to reflect the creation of SEMI E130.1.

1 Purpose

1.1 This document establishes a Prober Specific Equipment Model for prober equipment deployed in a factory adhering to the Global Joint Guidance for 300 mm standards (PSEM300). The PSEM300 consists of equipment characteristics and behaviors that apply to this class of equipment and are required to be implemented. The intent of this document is to facilitate the integration of prober equipment into an automated semiconductor factory. This document accomplishes this by defining an operational model for prober equipment as viewed by a host. This definition provides a standard host interface and equipment operational behavior.

2 Scope

2.1 The PSEM300 extends the generic Process State Model of SEMI E30 (GEM) with a Specific Process State model for prober equipment. This standard attempts to be protocol independent. There is no attempt to replace the protocol dependent aspects of SEMI E30 (GEM). Instead, this document assumes that certain equipment features not covered within the concepts and behaviors of the related 300 mm standards will be supported using the concepts and behaviors defined within SEMI E30. See Section 9 for a description of the SEMI E30 provisions.

2.2 This document does not attempt to define the full set of equipment capabilities. It is understood that additional value-added features will be provided in addition to the requirements set forth in this document.

2.3 This document intends host-instructed indexing of sites within a substrate and the loading/unloading of the substrates. Automated self-indexing with such communications as GPIB or RS-232C between tester and prober equipment is outside the scope of this specification but is not precluded. See the Related Information section for reference.

2.4 This document does not describe much about recovery of exception or manual operation to recover from alarm / potential-alarm (warning) situations. Such conditions are usually configuration specific.

2.5 This standard presents a solution from the concepts and behavior down to the messaging services. It does not define the messaging protocol.

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

3 Limitations

3.1 This document does not define the format of each data item specified in the equipment recipe.

3.2 This document does not define a communications protocol. In order to implement this specification, it requires a standard communications protocol and data format specification.

4 Referenced Standards

4.1 *SEMI Standards*

SEMI E5 — SEMI Equipment Communication Standard II Message Content (SECS II)

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

SEMI E40 — Standard for Processing Management

SEMI E42 — Recipe Management Standard: Concepts, Behavior, and Message Services

SEMI E87 — Specification for Carrier Management (CMS)

SEMI E90 — Specification for Substrate Tracking

SEMI E94 — Provisional Specification for Control Job Management

SEMI G81 — Specification for Map Data Items

SEMI G85 — Specification for Map Data Format

SEMI M20 — Specification for Establishing a Wafer Coordinate System

SEMI M21 — Specification for Assigning Addresses to Rectangular Elements in a Cartesian Array

4.2 *IEEE Standards*[1]

IEEE 754-1985 – IEEE Standard for Binary Floating-Point Arithmetic

4.3 *Other Documents*

Harel, D., "Statechart: A Visual Formalism for Complex Systems", Science of Computer Programming 8 (1987) 231-274.

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

5 Terminology

5.1 *Definitions*

5.1.1 *alignment* — a procedure in which a coordinate system is established on a substrate.

5.1.2 *cassette* — a physical object containing one or more substrate locations. See SEMI E87.

5.1.3 *chuck* — the primary stage for processing a substrate.

5.1.4 *column* — synonymous with the term "X-coordinate". Columns increase along the X axis.

5.1.5 *die* — 1. A field sub-unit. 2. An area of substrate that contains the device being manufactured.

5.1.6 *fiducial* – flat or notch in the physical substrate used to identify the substrate orientation.

5.1.7 *inker* — a resource of the prober. The electromechanical units to put ink mark on die.

5.1.8 *load* — 1. Move a substrate onto a substrate location. 2. Move a carrier onto the equipment.

5.1.9 *marking* — the process of the prober that deposits an ink mark on a die using the inker.

5.1.10 *pre-align* – set up a substrate to be processed on the chuck. The equipment may have a separate stage for performing the pre-align function.

5.1.11 *probe card* — the electromechanical interface necessary to enable temporary electrical contact between the substrate to be tested and the tester resource. May consist of multiple components.

5.1.12 *pipelin*e – an equipment configuration consisting of multiple stages through which the equipment sequences material in succession.

5.1.13 *row* — synonymous with the term "Y-coordinate". Rows increase along the Y axis.

5.1.14 *stage* – a general term for a substrate location that serves a specific function such as pre-align or chuck.

5.1.15 *state* — 1. A static set of conditions. If the conditions are met, the state is current (SEMI E30). 2. A state reacts predictably to specific stimuli.

5.1.16 *substrate* — 1. The basic unit of material, processed by PSEM300 equipment such as wafers.

5.1.17 *unload* — 1. Remove a substrate from a substrate location. 2. Remove a carrier from the equipment.

5.1.18 *X axis* — the X axis is the horizontal axis from left to right when the substrate is rotated according to Orientation.

5.1.19 *Y axis* — the Y axis is the vertical axis from bottom to top when the substrate is rotated according to Orientation.

1 Institute of Electrical and Electronics Engineers, IEEE Operations Center, 445 Hoes Lane, P.O. Box 1331, Piscataway, New Jersey 08855-1331, USA. Telephone: 732.981.0060; Fax: 732.981.1721, Website: www.ieee.org

6 Conventions

6.1 *Remote Command Parameter Table Definitions*

6.1.1 The table below provides an example of the tables used to list and describe arguments for services or remote commands defined in this specification.

Parameter	*Req/Ind*	*Rsp/Cnf*	*Form*	*Comment*
	(see below)	(see below)		

6.1.2 *Parameter* – Specifies the name of the parameter.

6.1.3 *Req/Ind* – Specifies relationship of the parameter for the command request or indication message. See below table for codes used.

6.1.4 *Rsp/Cnf* – Specifies relationship of the parameter for the command response or confirmation message. See below table for codes used.

6.1.5 *Form* – Specifies the data type of the parameter. Data types are as defined in the SEMI compilation of terms or are included in this specification. Specific representation of the data types will depend on and be defined by the communications protocol and data format adjunct specification.

6.1.6 The following codes appear in the Req/Ind and Rsp/Cnf columns to define how each parameter is used in each direction:

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

7 Requirements

7.1 Following sections in this document, except such supplementary descriptions as Related Information, are requirements of this specification.

8 State Model

8.1 The purpose is to define the equipment-specific processing state model to portray the expected operational states of the equipment to enable host tracking and control in place of a local operator. Process Job Management standard (SEMI E40) is part of the required behavior of this document as described in Section 9. For additional description of this behavior see Section 10 or See the actual standard, SEMI E40.

8.1.1 The Process State model tracks the state of the equipment with regard to its ability to perform processing on a substrate. The Process State model is used to inform the host when a substrate is loaded on the main chuck and has been properly setup for processing.

8.1.2 It is expected that the prober will be preparing a substrate to be loaded on the main chuck while the host is processing an existing substrate on the main chuck. This concurrent activity does not impact the host's ability to process the existing substrate and therefore does not cause a change in the Process State model. The standard for Substrate Tracking (SEMI E90) is used for tracking substrates between individual substrate locations.

8.1.3 The processing state model in this document is required for implementing a PSEM300-compliant prober, in addition to the required state models in the related standards. A state model consists of a state model diagram, state definitions, and a state transition table. A state model represents the host's view of the prober, but not necessarily the actual prober operations. All PSEM300 state model transitions shall be mapped sequentially into the actual

equipment events that satisfy the requirements of those transitions. In certain implementations, the prober may enter a state and has already satisfied all of the conditions required by the PSEM300 state model for transition to another state. In this situation, the prober makes the required transition without any additional actions.

8.1.4 Some equipment may need to include additional states. However, any additional states must not change the PSEM300-defined state transitions. All expected transitions between PSEM300 states must occur.

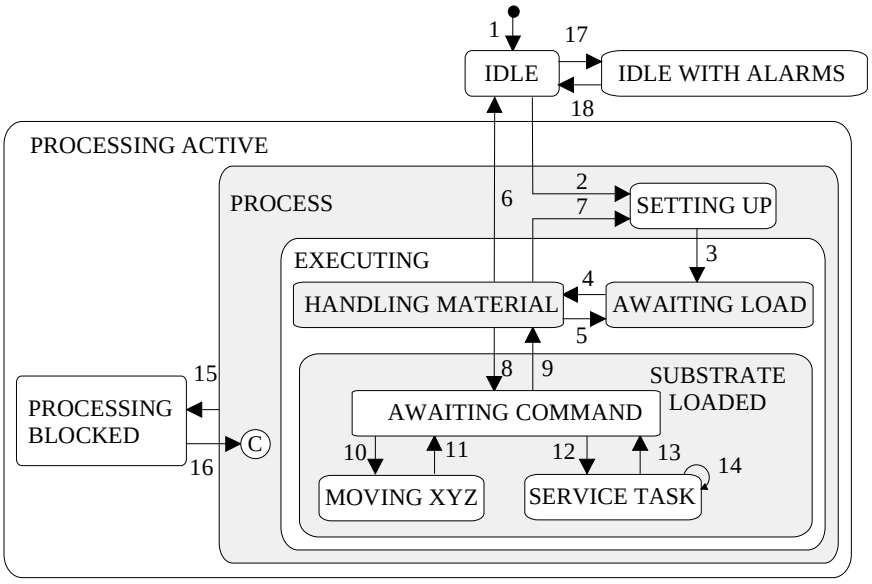

Figure 1
PSEM Process State Model Diagram

8.1.5 *Description of Prober Processing States*

8.1.5.1 *AWAITING COMMAND (SUBSTRATE LOADED Sub-state)* — A substrate on the main chuck is ready for processing. The prober is ready to receive an instruction from the host.

8.1.5.2 *AWAITING LOAD (EXECUTING Sub-state)* — At least one Process Job is in the PROCESSING state. No substrate is loaded on the main chuck. The prober is waiting for a host command to load a substrate.

8.1.5.3 *EXECUTING (PROCESS Sub-state)* — One of the following conditions exists: A) At least one Process Job is in the PROCESSING state and the prober is processing material as directed by the host. B) A single Process Job is active in either the STOPPING or ABORTING state and the prober is removing material.

8.1.5.4 *IDLE* — No Process Jobs are in the ACTIVE state and the processing resource is available without alarms.

8.1.5.5 *HANDLING MATERIAL (EXECUTING Sub-state)* — The prober is loading a substrate, unloading a substrate, or a combined action of loading and unloading substrates on the main chuck. When loading a substrate onto the main chuck, the HANDLING MATERIAL state is maintained until all preconditioning tasks (i.e. aligning, profiling, etc.) are completed and the substrate is ready for processing. If a condition arises where the substrate on a chuck must be re-aligned, the HANDLING MATERIAL state is active until the alignment task has completed.

8.1.5.6 *MOVING XYZ (SUBSTRATE LOADED Sub-state)* — The prober is executing a move command. This state is maintained until the move is complete. This includes motion in the z-axis. When this state is exited, the target unit(s) on the substrate is in position to be processed.

8.1.5.7 *PROCESS (PROCESSING ACTIVE Sub-state)* — This state is the parent of those sub-states that refer to the preparation and execution of a process job.

8.1.5.8 *PROCESSING ACTIVE* — The state in which at least one Process Job is active.

8.1.5.9 *PROCESSING BLOCKED* — The processing resource is not available for processing.

8.1.5.10 *SETTING UP (PROCESS Sub-state)* — A single ACTIVE Process Job is in the SETTING UP state.

8.1.5.11 *SERVICE TASK (SUBSTRATE LOADED Sub-state)* — A service task such as inking, cleaning, or inspection is being performed by the prober.

8.1.6 *PSEM Processing State Transition Table*

Table 1 PSEM Processing State Transition Table

#.	*Current State*	*Trigger*	*New State*	*Action*	*Comment*
1	No state	The processing resource has become available for processing.	IDLE		There are no active process jobs at this time.
2	IDLE	A process job has been dispatched into the SETTING UP state.	SETTING UP	Actions taken correspond to preconditioning activities within the SEMI E40 Process Job.	
3	SETTING UP	A single Process Job is active and has transitioned from the SETTING UP state to the PROCESSING state.	AWAITING LOAD		The prober is awaiting instructions from the host to load a substrate on to the main chuck. The Process Job PRMtlNameList contains the list of substrates to be processed.
4	AWAITING LOAD	Prober has received a command from the host to load a substrate on to the main chuck.	HANDLING MATERIAL	The equipment is loading a substrate on the main chuck.	None.
5	HANDLING MATERIAL	The material handling task completes leaving the main chuck unoccupied. A Process Job is in the PROCESSING state.	AWAITING LOAD		See Section 17 for a description of the prober behavior related to material handling.
6	HANDLING MATERIAL	The material handling task completes leaving the main chuck unoccupied. There are no active Process Jobs.	IDLE		
7	HANDLING MATERIAL	The material handling task completes leaving the main chuck unoccupied. There are no Process Jobs in the PROCESSING state. A Process Job is in the SETTING UP state.	SETTING UP		
8	HANDLING MATERIAL	The material handling task completes. A substrate located on the main chuck has been fully setup and is ready to receive processing as directed by the host.	AWAITING COMMAND		All pre-processing steps including profiling, aligning, temperature soak, etc. have completed. See Section 22.4 for the list of remote commands the host can issue while the prober is in the AWAITING COMMAND state.
9	AWAITING COMMAND	The host issued a material handling related remote command.	HANDLING MATERIAL	The prober performs the task as commanded by the host.	See Section 17 for a description of the commands the host may issue relating to material handling.

#.	Current State	Trigger	New State	Action	Comment
10	AWAITING COMMAND	The prober receives a motion related remote command.	MOVING XYZ	The prober moves the main chuck to the specified location on the substrate.	See Section 17 for a description of the motion related commands the host can issue to trigger this transition. See the Collection Event Table Section 19.2 for the list of required data items associated with this transition.
11	MOVING XYZ	The prober has completed the motion related task and waits for further instructions from the host.	AWAITING COMMAND		Prober waits for further instructions from the host.
12	AWAITING COMMAND	The prober received a remote command specifying a service task.	SERVICE TASK	See Section 16 for a description of the prober actions related to performing a service task.	
13	SERVICE TASK	The prober has completed a service task.	AWAITING COMMAND		Prober waits for further instructions from the host.
14	SERVICE TASK	The prober is configured to perform a subsequent service task upon completing a current service task.	SERVICE TASK	See Section 16 for a description of the prober actions related to performing a service task.	Example: The recipe may call for a probe card inspection to automatically follow a probe clean service task.
15	PROCESS	The processing resource has become unavailable.	PROCESSING BLOCKED	No further processing is possible.	The cause of this trigger can be an internal interrupt or an external command from the operator or host.
16	PROCESSING BLOCKED	The processing resource has become available or the prober is handling material in the process of terminating a job.	PROCESS Sub-state	See Section 8.1.7 for the conditional return state.	If the job(s) have been terminated, the HANDLING MATERIAL state will be occupied while the material is being removed from the prober.
17	IDLE	An alarm state is entered that causes the processing resource to be unavailable for processing.	IDLE WITH ALARMS		
18	IDLE WITH ALARMS	The alarm condition has been cleared.	IDLE		

8.1.7 *Process Model Conditional Table*

Table 2 Process Model Conditional Table

Conditions	Next State
The current Process Job is terminating	HANDLING MATERIAL
The processing resource has become available and the current Process Job remains active	Previous EXECUTING state

9 Related Standards Hierarchy

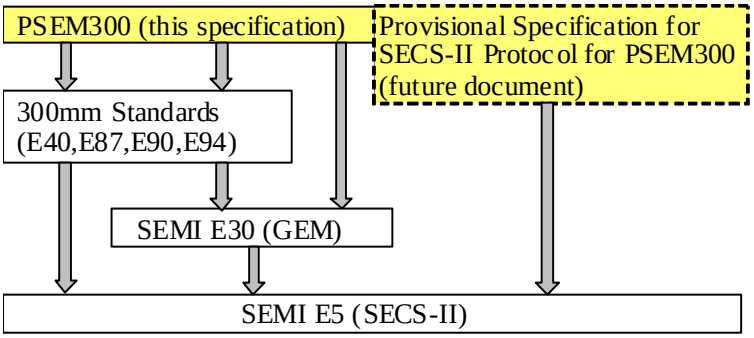

Figure 2
Related Standards Hierarchy

9.1 *SEMI E30 Relationship* — This document does not attempt to redefine the concepts and behaviors of SEMI E30 (GEM). Instead, this document takes an hierarchical approach to identifying the methods for achieving a given task as shown in Related Standards Hierarchy.

9.1.1 *SEMI E30 Inclusions* — All of the SEMI E30 state models are required with exception of the processing state diagram. This diagram is replaced with the one defined in this standard.

9.1.2 *SEMI E30 Exclusions* — Data Type restrictions may conflict with the newer 300 mm standards and therefore, the constraints found within the newer standards shall take precedence over the restrictions found in SEMI E30 section 5 “Data Types”.

10 Process Jobs

NOTE 1: SEMI E40 is required for PSEM300 compliance.

10.1 A PSEM300 prober is not able to operate as an independent processing agent. It requires a host to coordinate interaction with the tester. This document defines a suitable GEM interface consisting of the Process State Model and associated remote commands to allow the host to act as the processing agent on behalf of the prober to complete the execution of the process job.

10.2 This standard assumes that if the equipment has a pipeline (e.g. pre-align stage and main chuck), the equipment will support concurrent process jobs such that when the last substrate of an active process job leaves the first pipeline stage, the equipment will launch another process job thus maintaining continuous material flow.

10.3 *Process Job Preconditioning* — This standard does not attempt to define the set of recipe attributes that, when different from the current equipment configuration, will require a transition to the Process Job SETTING UP state (preconditioning required). The recipe attributes that do impact the Process Job transition to SETTING UP will be user defined and documented as such.

10.4 A PM compliance checklist must be filled out to indicate the level of compliance to both the Fundamental and Additional PM capabilities.

11 Control Job

NOTE 2: SEMI E94 is required for PSEM300 compliance.

11.1 See SEMI E94 for details of Control Job management.

11.2 A substrate’s destination substrate location is specified in the Control Job’s MtrlOutSpec attribute. If the MtrlOutSpec attribute is not set, the destination substrate location defaults to the source substrate location.

11.3 A CJM compliance checklist must be filled out to indicate the level of compliance to Fundamental CJM capabilities (no Additional capabilities are defined at this time).

12 Carrier Management

NOTE 3: SEMI E87 is required for PSEM300 compliance.

12.1 *CarrierReCreate* — After processing material for a given carrier, the host must be able to define a new CJ/PJ for the same carrier without having to remove the carrier from the tool. For this, the equipment must support the CarrierReCreate service.

12.2 A CMS compliance checklist must be filled out to indicate the level of support for CMS for a given equipment configuration.

13 Substrate Tracking

NOTE 4: SEMI E90 is required for PSEM300 compliance.

13.1 Substrate Tracking must be supported for all product substrates and substrate locations where product substrates may reside. Some of the substrate locations that might exist include: handling robot, post-chuck substrate location.

13.2 The supplier may choose not to support substrate tracking for certain internal substrate locations. This is allowable if the substrate location will not be occupied by product substrates (e.g. Cleaning Media tray, AUX Tray). The supplier must document which substrate locations are available for substrate tracking.

13.3 A CMS "CarrierReCreate" service will cause each of the substrates contained within a recreated carrier to be reset to the AT SOURCE and NEEDS PROCESSING state.

NOTE 5: A STS compliance checklist must be filled out to indicate the level of compliance to STS Fundamental and Additional capabilities.

14 Equipment Configuration

14.1 This standard does not attempt to specify the physical equipment configuration. The intent of this standard is to provide a generalized specification that applies to all configurations. The supplier is responsible for assuring that additional features integrate consistently with the relevant standards impacted by the feature. The purpose of this section is to add clarification to baseline assumptions that are implied throughout this specification.

14.2 *Pipeline*

14.2.1 A typical pipeline configuration consists of a pre-align stage used to pre-process a substrate before loading onto the chuck. The equipment can begin processing the substrate on the chuck while the next substrate is being prepared on the pre-align stage. For this configuration, the material flow is assumed to follow the path: source substrate location ⇨ pre-align stage ⇨ chuck ⇨ destination substrate location. There may be additional substrate locations both before and after the chuck.

15 Control Maps

15.1 Control Maps provide a mechanism for communicating the selective sites for processing in a pre-determined manner. Control Maps are not suitable for adaptive test where dynamic site selection is used.

15.2 *Control Map Uses*

15.2.1 *Inking* — The host can download a control map containing the sites to be inked. The host issues the INK DIE remote command specifying the control map as the source for the die list.

15.2.2 *Normal Processing* — When using a Control Map, the host issues the INDEX remote command to move the prober coordinates to the next site listed in the Control Map. The host may issue a move related command other than INDEX to move the chuck to any location on the substrate. Move commands other than INDEX, do not impact the state of the Control Map. The prober processes subsequent INDEX commands by going to the next site listed in the Control Map. If the Control Map does not contain additional sites the equipment sends an event to the host indicating that the control map has been completed.

15.2.3 *Interrupted Processing* — If the processing of a substrate using a Control Map is interrupted (i.e. Correlation Test, Probe Cleaning with a cleaning substrate, Chuck Reset, etc...), the equipment shall resume processing with the Control Map from the point where processing was suspended when the substrate processing resumes. The exceptions to this requirement are

15.2.3.1 If the current Process Job is altered such that the suspended substrate is no longer in the IN PROCESS state, the Control Map is no longer valid.

15.2.3.2 If the host issues a RESET-CONTROL-MAP command, the Control Map will resume at the first site. If the host designates coordinates with the RESET-CONTROL-MAP command, the prober will resume the Control Map with the designated coordinates. It is the responsibility of the host to assure that the coordinates provided with this command are contained in the Control Map.

15.3 If the host issues a SET-CONTROL-MAP command, the prober will begin processing the Control Map at the first site of the designated Control Map. If the host designates coordinates with the SET-CONTROL-MAP command, the prober will process the Control Map beginning with the designated coordinates. It is the responsibility of the host to assure that the coordinates provided with this command are contained in the Control Map.

15.4 *Control Map Transmission* — The method for downloading/uploading control maps is outside the scope of this document.

15.5 *Control Map Structure* — The structure of the control map is outside the scope of this document. The use of SEMI G81 or SEMI G85 standards is recommended to describe map structures.

15.6 *Control Map Modifications* — The option to update a control map to register sites as they are processed is a user specific detail. In the event that control maps are modified on the prober, the prober must support uploading the modified maps to the host.

16 Service Tasks

16.1 The recipe on a PSEM300 compliant prober contains setup parameters and references to a collection of service recipes. (See SEMI E42 for a description of Service Recipe's). The host can issue a command to execute a service task. The instructions for performing the specified task are contained in the respective service recipe. This standard makes no attempt to define the full list of service tasks available for a given equipment. However, when a service task is available, the supplier is responsible for documenting the command and parameters required for the host to request the task. When a service task is performed, the Process State model transitions to the SERVICE TASK state and provides the host with a descriptive TaskID.

16.2 *Inking*

16.2.1 The host can perform inking on individual devices by issuing the INK-DEVICE command, or the host can ink a list of devices by issuing the INK-DIE command and a die list. The die list parameter may reference a Control Map containing the die selection.

16.3 *Inspecting*

16.3.1 There are several inspections the prober may be capable of performing such as inspecting the probes, inspecting ink marks, and inspecting the Probe to Pad alignment. The supplier must document the inspections that are supported and provide remote commands for selecting the designated inspection.

16.3.2 The Inspection associated with the maintenance activity is not to be confused with the process of Inspecting as defined in this section. The activity of inspecting the equipment in a maintenance capacity may involve the use of subsystems including one or more inspection process steps. While the use of an inspection subsystem in a process capacity constitutes a transition to the SERVICE TASK state, the equivalent use of the same subsystem in a maintenance capacity does not.

16.4 *Probe Cleaning*

16.4.1 This document does not attempt to specify the full set of probe cleaning features. However, this document does require at minimum the ability to designate logical zones within a cleaning media type to prevent cross-contamination between wafer technology types. The supplier must provide suitable means through the use of Equipment Constants and Status Variables to allow the user full selectivity of the available options.

17 Material Movement

17.1 *Between Substrate Locations (Inter-Substrate Moves)*

17.1.1 *Loading Substrates*

17.1.1.1 The host issues the LOAD-SUBSTRATE command to load the next substrate identified in the current Process Job's PRMtlNameList attribute onto the chuck. The order by which the substrates of a process job are processed is fully defined within the Process Job.

17.1.1.2 If the equipment is configured with a multi-stage pipeline, upon receiving the LOAD-SUBSTRATE command from the host, the equipment will fill the pipeline up to the point where the substrate to be processed is loaded onto the chuck and readied for processing. Subsequent LOAD-SUBSTRATE commands will maintain a full pipeline as long as additional substrates are available for processing. This assures continuous material flow.

17.1.1.2.1 Example: if the equipment is configured with a two-stage pipeline consisting of a pre-align stage and a main chuck, A LOAD-SUBSTRATE command issued by the host while the Process State model is in the AWAITING LOAD state and the pipeline is empty will cause the first substrate to be processed through the pre-align stage and onto the chuck. The next substrate will then be loaded onto the pre-align stage.

17.1.2 *Loading Substrates—Spanning Process Job Boundaries*

17.1.2.1 When the host issues the LOAD-SUBSTRATE command that moves the last substrate of the current Process Job out of the first pipeline stage (or chuck for non-pipelined configurations) and another Process Job is defined with the PRProcessStart attribute set to TRUE, the equipment will automatically launch the next Process Job. If Process Job preconditioning is not required for the newly launched Process Job, the first substrate listed in the PRMtlNameList will be loaded into the first pipeline stage (or chuck for non-pipelined configurations).

17.1.3 *Loading Substrates—Substrate State Model*

17.1.3.1 Substrate Tracking events (SEMI E90) must be available to the host. When a substrate is moved from the source carrier to the first pipeline stage (or chuck for non-pipelined configurations), the substrate state model will transition from AT SOURCE to AT WORK.

17.1.3.2 When a substrate is loaded on the chuck, the substrate state model will transition from NEEDS PROCESSING to IN PROCESS for substrates that were not previously suspended (see Section 17.1.5 for

a description of suspending substrates). A substrate that was suspended will already be in the IN PROCESS state.

17.1.4 *Unloading Substrates*

17.1.4.1 *LOAD-SUBSTRATE Command* — When the host issues the LOAD-SUBSTRATE command, the substrate currently occupying the chuck is removed from the chuck and placed in the destination substrate location (This assumes that no additional post-chuck substrate locations exist). The substrate state model transitions from the IN PROCESS to PROCESSED state. When the substrate is loaded in its destination substrate location, the substrate transitions from the AT WORK to the AT DESTINATION state.

17.1.4.2 *UNLOAD-SUBSTRATE Command* — The host issues the UNLOAD-SUBSTRATE command to remove the substrate from the chuck and place it in its destination substrate location leaving the chuck unoccupied. For pipelined configurations, the substrate(s) in the pipeline remain in their current location. The state model for the substrate removed from the chuck transitions from the IN PROCESS to PROCESSED state. When the substrate is loaded in its destination substrate location, the substrate transitions from the AT WORK to the AT DESTINATION state. This command cannot be used to suspend the substrate occupying the chuck.

17.1.4.3 *UNLOAD-ALLSUBSTRATES Command* — The host issues the UNLOAD-ALLSUBSTRATES command to return all substrates occupying the chuck and pipeline stages to their respective carriers. For pipelined configurations, substrates occupying a pre-chuck pipeline stage are returned to their source substrate location. The state model for these substrates transitions from the AT WORK to AT SOURCE state while remaining in the NEEDS PROCESSING state.

17.1.4.3.1 *UNLOAD-ALLSUBSTRATES with SUSPEND* — If the host intends to continue processing the substrate occupying the chuck, the host includes the SUSPEND parameter with the UNLOAD-ALLSUBSTRATES command. (See Section 17.1.5 for a description of suspending substrates.)

17.1.4.3.2 *UNLOAD-ALLSUBSTRATES no SUSPEND* — If the host has completed processing the substrate occupying the chuck, the host issues the UNLOAD-ALLSUBSTRATES without the SUSPEND parameter. The substrate occupying the chuck transitions from the IN PROCESS to PROCESSED state and is delivered to the destination substrate location. When the substrate is loaded in the destination substrate location, the substrate transitions from the AT WORK to the AT DESTINATION state.

17.1.5 *Suspending Substrates*

NOTE 6: Substrate Suspending and Resuming are not Job Control collection events (e.g. CJ/PJ PAUSE/RESUME). A substrate can be suspended as part of a normal process interrupt to allow a service task to be performed as directed by the host.

17.1.5.1 The host can suspend processing on a substrate in order to free the chuck for other purposes such as performing diagnostics or maintenance. A suspended substrate will be returned to its source substrate location and remains in the IN PROCESS state. When the substrate is loaded into the source substrate location, the substrate transitions from the AT WORK to the AT SOURCE state.

17.1.5.2 Suspending a substrate does not change the order in which substrates are processed within a Process Job.

17.1.5.3 The commands that result in a substrate being suspended are: UNLOAD-ALLSUBSTRATES with SUSPEND and LOAD-TEST-SUBSTRATE. It is expected that additional capabilities may exist that result in a substrate being suspended (such as cleaning probes with a cleaning disk).

17.1.6 *Resuming a Suspended Substrate*

17.1.6.1 A substrate that has been suspended will be the first substrate loaded onto the chuck when processing resumes.

17.1.6.2 If a substrate was suspended automatically as part of a service task, the equipment will automatically resume the suspended substrate prior to indicating that the service task has completed.

17.1.6.3 If the host suspended a substrate by issuing the UNLOAD-ALLSUBSTRATES with SUSPEND, the host resumes the suspended substrate by issuing the LOAD-SUBSTRATE command.

17.2 *Chuck Movement (Intra-Substrate Moves)*

17.2.1 When a control map is used to specify stepping, the INDEX Remote command must be used to move to the next die location. When a control map is not used, either the MOVE, MOVE-ABS or MOVE-SUBSTEP Remote commands must be used to specify motion.

17.2.2 Die coordinates are row and column numbers. Moving the wafer one row increases or decreases the DieYCoordinate by one. Moving the wafer one column increases or decreases the DieXCoordinate by one. Following an INDEX command the die coordinates can be retrieved from the variables “DieXCoordinate” and “DieYCoordinate”.

17.2.3 The Z coordinate is zero if the wafer is out of contact (the chuck is down) with the probes and one if it is in contact (the chuck is up).

17.2.4 The MOVE command specifies the number of rows and columns to move from the current position. When the move is complete, DieXCoordinate will be changed to DieXCoordinate+X; DieYCoordinate will be changed to DieYCoordinate+Y. When the Z parameter is included in the MOVE command, a value of (Z = –1) moves the chuck to the down coordinate, a value of (Z = +1) moves the chuck to the up coordinate, and a value of (Z = 0) leaves the chuck in the starting Z coordinate.

17.2.5 The MOVE-ABS command specifies the new die coordinates. When the move is complete, DieXCoordinate will be changed to X; DieYCoordinate will be changed to Y; and the chuck will be down (Z = 0) or up (Z = 1).

17.2.6 The MOVE-SUBSTEP remote command moves the substrate in units smaller than the die size. The step resolution for the X,Y and Z axis is available for query from the equipment status variables MinXStep, MinYStep and MinZStep. X, Y, and Z are all relative motions in the specified units. The prober may apply an appropriate compensation factor to adjust for thermal expansion, etc.

18 Coordinate System

18.1 This standard provides a coordinate system capable of describing simple geometries. Additional user-defined attributes may be required in addition to the attribute definitions contained in this specification.

18.2 This coordinate system does not include information for establishing which devices on a substrate are “good” devices. The guidelines for assigning the row and column index number are set forth in SEMI G81.

18.3 *General Coordinate Attributes*

18.3.1 *Rows* — The number of rows on a substrate in the Y-axis. A row must contain at least one complete device to be included in this count.

18.3.2 *Columns* — The number of columns on a substrate in the X-axis. A column must contain at least one complete device to be included in this count.

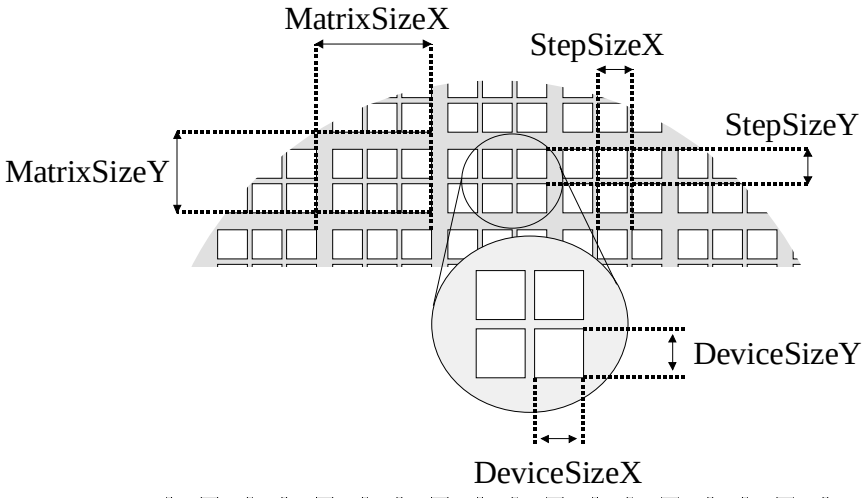

Figure 3
Dimensional Attributes

18.4 *Matrix Substrates*

18.4.1 The term “Matrix” is used to describe the step periodicity associated with an exposure field. Applicable when the step size between adjacent devices spanning an exposure field is different from the step size of adjacent devices within an exposure field.

18.4.2 This standard does not address complex geometries where individual device sizes vary within an exposure field or between exposure fields or where exposure step sizes vary across the substrate.

18.4.3 A non-matrix substrate is a substrate where the step size between adjacent devices spanning an exposure field is no different from the step size between adjacent devices within an exposure field.

18.4.4 *Matrix Attributes*

18.4.4.1 *MatrixDeviceRows* — This is the number of device rows within an exposure field. This value is dependent upon the Orientation recipe attribute but not the OrientationOnChuck attribute. For non-matrix substrates, the value of this attribute is 1.

18.4.4.2 *MatrixDeviceCols* — This is the number of device columns within an exposure field. This value is dependent upon the Orientation recipe attribute but not on the OrientationOnChuck attribute. For non-matrix substrates, the value of this attribute is 1.

18.4.4.3 *MatrixSizeX* — The X-axis step size in microns between a device in one exposure field and the same device in an adjacent exposure field. For non-matrix substrates, the value of this attribute is equal to the StepSizeX attribute. This attribute is dependant on the Orientation recipe attribute.

18.4.4.4 *MatrixSizeY* — The Y-axis step size in microns between a device in one exposure field and the same device in an adjacent exposure field. For non-matrix substrates, the value of this attribute is equal to the StepSizeY attribute. This attribute is dependant on the Orientation recipe attribute.

18.5 *Reference Device*

18.5.1 The Reference Device establishes the link between the coordinate system Row and Column and the physical location on the substrate.

18.5.2 *Reference Device Attributes*

18.5.2.1 *RefDevicePosX* — This is the offset of the center of the reference device in the X direction from the center of the substrate with the substrate Orientation at 0º and OriginLocation = 3.

18.5.2.2 *RefDevicePosY* — This is the offset of the center of the reference device in the Y direction from the center of the substrate with the substrate Orientation at 0º and OriginLocation = 3.

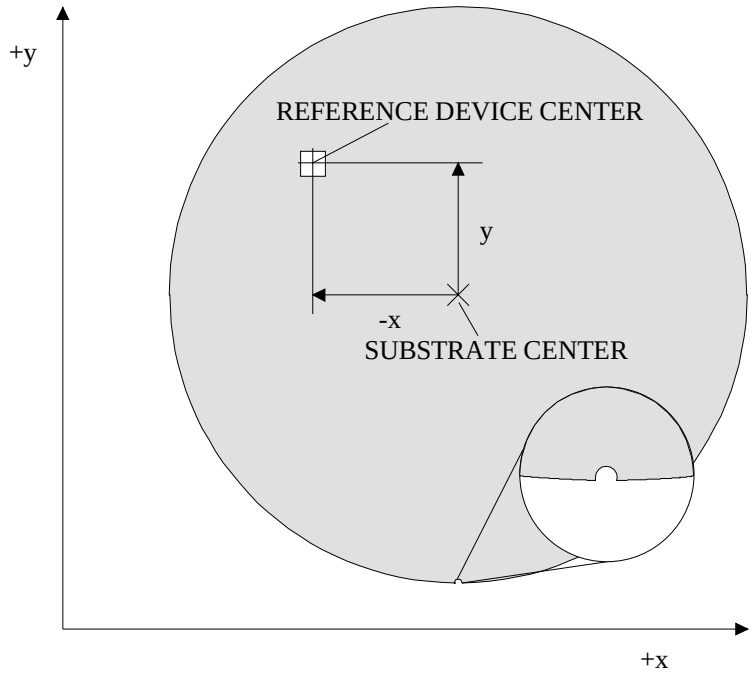

Figure 4
Reference Device Position (RefDevicePosX and RefDevicePosY)

18.5.2.3 *ReferenceDeviceX* — This is the Column number corresponding to the Reference Device. This value is dependent upon the Orientation and OriginLocation recipe attributes but not the OrientationOnChuck attribute.

18.5.2.4 *ReferenceDeviceY* — This is the Row number corresponding to the Reference Device. This value is dependent upon the Orientation and OriginLocation recipe attributes but not the OrientationOnChuck attribute.

18.5.2.5 *RefDeviceMatrixX* — This is the Column number corresponding to the Reference Device within a given exposure field. This value is dependent upon the Orientation and OriginLocation recipe attributes but not the OrientationOnChuck attribute. For non-matrix substrates, this attribute is 1.

18.5.2.6 *RefDeviceMatrixY* — This is the Row number corresponding to the Reference Device within a given exposure field. This value is dependent upon the Orientation and OriginLocation recipe attributes but not the OrientationOnChuck attribute. For non-matrix substrates, this attribute is 1.

18.6 *Coordinate Origin* — The Coordinate Origin identifies the axis quadrant occupied by the substrate for addressing devices on the substrate.

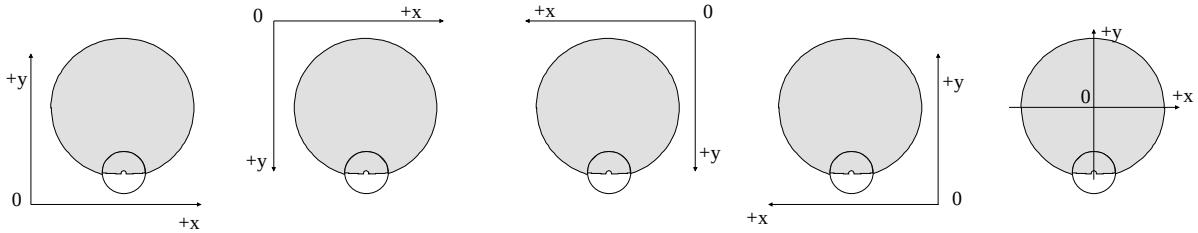

Figure 5
Coordinate Origin (OriginLocation)

18.6.1 *Origin Recipe Attributes*

18.6.1.1 *OriginLocation* — Coordinate system on substrate to address devices. Enumerated:

0 = Center, 1 = Upper Right, 2 = Upper Left 3 = Lower Left (default), 4 = Lower Right.

18.7 *Orientation*

18.7.1 Orientation addresses the location of the substrate primary fiducial (flat or notch).

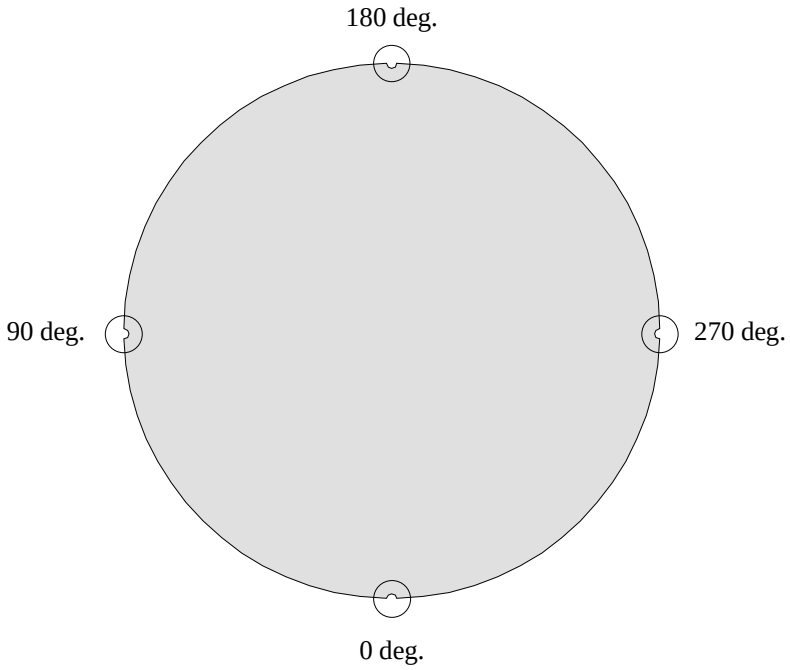

Figure 6
Orientation of the Substrate Notch

18.8 *Orientation Attributes*

18.8.1 *Orientation* — Identifies the orientation of the fiducial when defining the coordinate system on a substrate. This parameter in conjunction with the OriginLocation recipe attribute determines the coordinate and size attributes of devices on a substrate.

18.8.2 *OrientationOnChuck* — Identifies the orientation of a substrate loaded on the chuck. This attribute does not affect the coordinate and size attributes of devices on a substrate.

18.9 *Example Coordinate References* — The values presented in this section are for demonstration purposes (measurements are not exact).

18.9.1 *Orientation* — The recipe attributes presented in Figure 7 reflect an Orientation attribute setting of 0º. The recipe attributes presented in Figure 8 reflect an Orientation attribute setting of 90º.