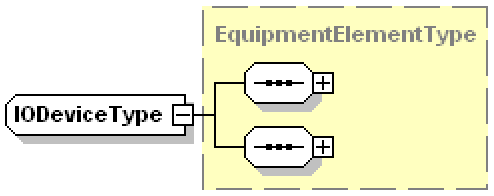

Figure 11
IODeviceType

6.3.9 *MaterialLocation => MaterialLocationType*

6.3.9.1 The MaterialLocation class is mapped to the XML MaterialLocationType. MaterialLocationType extends NameableType (see Figure 12).

6.3.9.2 Table 11 shows how the attribute of the MaterialLocation class is mapped into XML.

Table 11 Translation Table For MaterialLocation

Attribute or Role Name	*UML Type*	*XML Element or Attribute (or Reference)*	*XML Name/Type*
materialType	enumeration	element	MaterialType: MaterialTypeEnum

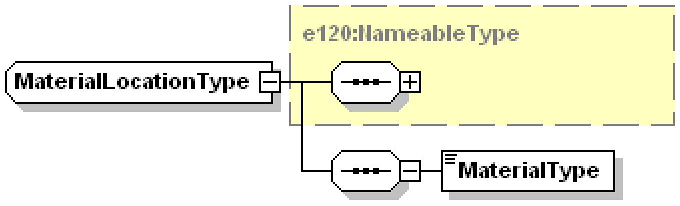

Figure 12
MaterialLocationType

6.3.10 *SoftwareModule => SoftwareModuleType*

6.3.10.1 The SoftwareModule class is mapped to the XML SoftwareModuleType. Table 12 shows how the attributes of SoftwareModule are mapped to XML. The resulting XML structure is illustrated in Figure 13.

Table 12 Translation Table For SoftwareModule

Attribute or Role Name	*UML Type*	*XML Element or Attribute (or Reference)*	*XML Name/Type*
name	string	element	Name: xs:string
supplier	string	element	Supplier: xs:string
description	string	element	Description: xs:string
version	string	element	Version: xs:string

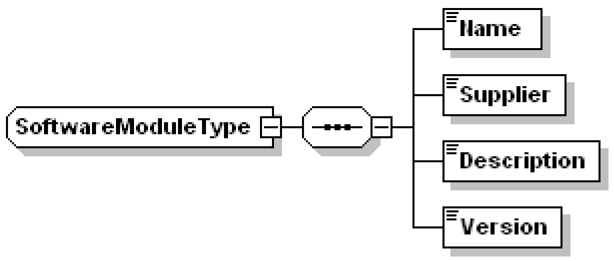

Figure 13
SoftwareModuleType

6.4 *Added XML Constructs*

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

6.4.2 *XML simpleTypes*

6.4.2.1 Table 13 describes each simpleType[9] contained in the XML schema.

Table 13 simpleTypes In The CEM Schema

Type	*Description*
UuidType	Pattern for 36-character uuid string is defined.
LocatorType	String pattern defined for the Locator value as defined in the Related Information section of SEMI E120 This is included to promote common definition of Locator. Its use is not required by SEMI E120 or this specification.
NameType	String pattern is defined, limited to certain characters as defined in SEMI E120.
MaterialTypeEnum	Defines possible values “Carrier”, “ProcessDurable”, and “Substrate” for MaterialLocationType.
ProcessTypeEnum	Defines possible values “Measurement”, “Process”, “Storage”, and “Transport” for ExecutionElementType.

6.4.3 *Added XML complexTypes*

6.4.3.1 Table 14 describes each added complexType[10] contained in the XML schema that was not described above.

Table 14 complexTypes In The CEM Schema

Type	*Description*
ExtensionArrayType	Array of ExtensionType associated with NameableType.
SoftwareModuleArrayType	Array of SoftwareModuleType aggregated by EquipmentElementType.
MaterialLocationArrayType	Array of MaterialLocationType aggregated by EquipmentComponentsType, ModuleComponentsType, and SubsystemComponentsType.
IODeviceArrayType	Array of IODeviceType aggregated by EquipmentComponentsType, ModuleComponentsType, and SubsystemComponentsType.
IODeviceRefArrayType	Array of IODeviceRef (UuidType) aggregated by EquipmentComponentsType, ModuleComponentsType, and SubsystemComponentsType.
SubsystemArrayType	Array of SubsystemType aggregated by EquipmentComponentsType, ModuleComponentsType, and SubsystemComponentsType.
SubsystemRefArrayType	Array of SubsystemRef (UuidType) aggregated by EquipmentComponentsType, ModuleComponentsType, and SubsystemComponentsType.

9 “simpleType” is an XML term (see also “xs:simpleType”).
10 “complexType” is an XML term (see also “xs:complexType”).

6.4.4 *Equipment Element*

6.4.4.1 Equipment is an element of type EquipmentType (see Figure 14). The Equipment element is included in the schema to provide for better type checking through the use of Keys and Keyrefs.

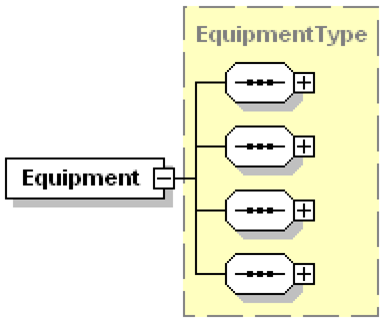

Figure 14
Equipment Element

6.4.4.2 The key and keyref constructs in the schema allow the enforcement the structure of the document. A key defines an element that may be used to uniquely identify a structure in the document. To uniquely identify the structure in the document, the key element must remain unique within the document. A document will fail validation if an instance of the element is found that is not unique.

6.4.4.3 The keyref construct allows a reference to a specific key to be created. The keyref requires an instance of the element exist with the same value as the reference. The key and keyref constructs allow the enforcement of the Uid elements in the CEM schema. A key exists for all Uid elements in the document forcing each Uid defined in the document to be unique and creates a table of them. This key is then used to enforce references to all objects base on Nameable in the document. Additional keys and keyref fields are defined to enforce a reference to an IODevice, Subsystem, or Module.

6.4.4.4 The unique construct is also used to enforce the structure of the document. The unique construct defines an element that must be unique within a portion of the document. For the CEM schema, the unique construct enforces the uniqueness of object names within an aggregation. The enforcement of names is limited to Nameable objects defined directly within the aggregation. It does not enforce names of objects in references.

7 XML Schema

7.1 The XML schema defined by this specification is contained in a separate document named:

E120-1.V0704.CommonEquipmentModel.xsd

7.2 That schema document is a part of this specification and it should accompany this document.

7.3 The contents of the schema document constitute the core part of this specification. The information above serves only to explain and describe the schema.

8 Related Documents

8.1 None

SEMI E121-0305
GUIDE FOR STYLE & USAGE OF XML FOR SEMICONDUCTOR MANUFACTURING APPLICATIONS

This guide was technically approved by the Global Information and Control Committee and is the direct responsibility of the North American Information and Control Committee. Current edition approved by the North American Regional Standards Committee on November 4, 2004 and December 10, 2004. Initially available at www.semi.org February 2005; to be published March 2005. Originally published March 2003; last published March 2004.

1 Purpose

1.1 The inclusion of XML (Extensible Markup Language) notation within current SEMI standards reflects the industry's growing interest in XML technology. This guide is the first step to prepare for XML protocols in SEMI standards and to establish consistency in XML style and usage. Recommendations given in this document are intended for the definition and conformance of XML based standards. This document also attempts to highlight those areas where standardization and further work may be required.

2 Scope

2.1 This guide is intended to promote interoperable application of XML in a variety of semiconductor manufacturing usage contexts. It addresses an approach for the usage and definition of XML specifications within the semiconductor industry with the focus on the use of XML in future SEMI standards. These guidelines should be applied to any XML document or schema generated as part of a SEMI standard for communication between software entities. This guide includes the following:

2.1.1 Overview of XML Resources, Organizations, and Standards,

2.1.2 XML Three Tier Model Architecture, and

2.1.3 General and specific recommendations of XML style and usage.

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 Because of the rapid development of new standards in the XML arena at the time of the writing of this document, it was found that there are areas in XML where standards are still in the development stage. Some examples of emerging XML technologies are: XMI (XML Metadata Interchange), SOAP (Simple Object Access Protocol), and Web Services Routing. Therefore, this guide does not include recommendations in those areas until the governing bodies approve these specifications. A list of some of the areas where guidance may be necessary is provided below:

3.1.1 (UML) (Unified Modeling Language) models with derived schema definition

3.1.2 XMI (XML Metadata Interchange) usage and recommendations

3.1.3 XML Namespace definitions

3.1.4 XSLT (Extensible Stylesheet Language Transformations) usage and conversion rules

3.1.5 WSDL (Web Service Definition Language) usage and recommendations

3.1.6 SOAP (Simple Object Access Protocol) and transport recommendations

4 Referenced Standards

4.1 *Other References*

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

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

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

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

5 Terminology

5.1 *Abbreviations and Acronyms*

5.1.1 *DTD* — Document Type Definition

5.1.2 *HTTP* — Hypertext Transfer Protocol

5.1.3 *SGML* — Standard Generalized Markup Language

5.1.4 *SOAP* — Simple Object Access Protocol

5.1.5 *W3C* — World Wide Web Consortium

5.1.6 *WSDL* — Web Services Definition Language

5.1.7 *WWW* — World Wide Web

5.1.8 *XMI* — XML Metadata Interchange

5.1.9 *XML* — Extensible Markup Language

5.1.10 *XPATH* — XML Path Language

5.1.11 *XSL* — Extensible Stylesheet Language

5.1.12 *XSLT* — Extensible Stylesheet Language Transformations

6 Overview of XML Resources, Organizations, and Standards

6.1 There are a large number of resources in the area of XML. Most of the information is readily available from organizations such as OMG, W3C, RosettaNet, ebXML, OASIS, Sun Micro Systems, IBM and other through their web sites. There has been an extensive amount work in standardizing dictionaries and vocabularies for several industries. These documents can be used until further standards and guides specific to this industry are created.

6.2 *Resources*

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

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

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

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

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

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

6.2.7 RFC 2396 *Uniform Resource Identifiers (URI): Generic Syntax*. T. Berners-Lee, R. Fielding, L. Masinter. August 1998.

6.2.8 *Hyper Text Transfer Protocol 1.1* (HTTP, http://www.w3.org/Protocols/rfc2616/rfc2616.html)

6.2.9 *Simple Object Access Protocol 1.1* — W3C Note, 8 May 2000 (SOAP, http://www.w3.org/TR/SOAP)

6.2.10 *Web Service Definition Language 1.1* — W3C Note, 15 March 2001 (WSDL, http://www.w3.org/TR/wsdl)

6.2.11 ISO/IEC 11179 (1-6) Information Technology — Standardization and Specification of Data Elements

7 XML Three Tier Model Architecture

7.1 The following three-tier schema model is recommended:

- Primary Schemas (common SEMI models)
- Secondary Schemas (individual SEMI models)
- Tertiary Schemas (non SEMI models)

7.1.1 *Primary Schemas* — Primary Schemas are schemas that have been defined by SEMI. These schemas usually cover a particular area such as equipment, transport system, scheduler/dispatcher, maintenance, etc.

7.1.2 *Secondary Schemas* — Secondary Schemas are schemas that have been defined by SEMI. These schemas are for a particular functionality required by the application such as carrier management, process job, data collection, etc. Secondary Schemas may derive from Primary Schemas as a base by extension or restriction.

7.1.3 *Tertiary Schemas* — Tertiary Schemas are defined by the supplier of the tool or application. These Schemas may derive from either Primary or Secondary Schemas as a base by extension or restriction.

8 XML Recommendations

8.1 *XML Document Compliance*

8.1.1 All XML documents, by definition, must comply with the W3C Recommendation, XML version 1.0 (see http://www.w3.org/TR/REC-xml for more detail on the specification). The purpose of this compliance is to assure that all XML documents are "well-formed" and "valid". A "well-formed" document means that there is exactly one root element, all sub-elements (and recursive sub-elements) have delimiting start- and end-tags, and that they are properly nested within each other. A "well-formed" document also conforms to the XML syntax specification and does not include external references unless a schema reference is provided. An XML document is "valid" if it has an associated schema and complies with the constraints expressed in the schema. "Valid" XML data is also "well-formed".

8.2 *Use of XML Schema versus DTD*

8.2.1 All XML documents should conform to a specified set of production rules. These production rules should be specified as XML Schemas. XML Schemas are the preferable way to define XML documents. XML Schemas are modular in structure and can be built from multiple components (files) by including or importing the components.

8.3 *Using Elements versus Attributes*

8.3.1 When defining attributes from an object model, it is preferable to define them as XML elements such that they can be easily extended by other applications.

8.3.2 Use elements if the information is hierarchical, contains a content model, or could potentially become more complex or hierarchical in the future.

8.3.3 Use attributes when the information is metadata about the element.

8.4 *Naming Conventions*

8.4.1 The following list is a compendium of recommended practices to be used when naming attributes and simple or complex elements.

8.4.1.1 Use the UpperCamelCase convention for elements and the lowerCamelCase convention for attributes.

8.4.1.2 Do not use any punctuation in the element name; i.e. underscores, dashes, etc. Use only alphanumeric characters.

8.4.1.3 Acronyms should be all in uppercase. Following a Three (or more) Letter Acronym (TLA), all in uppercase, the next letter is lowercase, i.e.: TLAlowercase, not TLAUppercase.

8.5 *Common XML Structures*

8.5.1 When defining a list of elements where each has the same element name, nest the list of elements inside a parent element. Form the parent's element name by concatenating the name of the list elements with the word "List".

8.5.1.1 For example:

```
<SubstrateIDList>
    <SubstrateID>Substrate1</SubstrateID>
    <SubstrateID>Substrate2</SubstrateID>
    :
    <SubstrateID>Substrate25</SubstrateID>
</SubstrateIDList>
```

8.6 *Schema Element Organization* — The following recommendations apply only to the Primary and Secondary Schemas.

8.6.1 With the purpose of readability of the schema and to aid in the understanding of the XML document, the following guidelines are recommended when elements are defined in the XML document or schema:

8.6.1.1 Elements of type ID or KEY should be listed at the top of the schema, and elements of type IDREF or KEYREF should be listed second.

8.6.1.2 Elements of simpleType should be declared first, followed by elements of complexType.

8.7 *XML Namespaces*

8.7.1 A namespace name is a URI (Universal Resource Identifier) that uniquely identifies the namespace. A namespace is a group of names usually with a related purpose and context. The namespace is a globally unique name. This is ensured by using a domain names as part of the namespace definition or a pre-determined identifier. SEMI documents will use the URN (Universal Resource Naming) convention to define namespaces, suppliers or any other 3rd party suppliers can use either URN or URL (Universal Resource Locator) descriptions.

8.7.2 Each application or document definition is responsible for providing a namespace for the extension of its vocabulary definition. A namespace must be unique and to ensure its uniqueness the definitions of namespaces must follow the rules described below.

8.7.3 *Namespace Syntax*

8.7.3.1 SEMI Namespaces use Uniform Resource Names (URNs). All URNs have the following syntax:

```
<URN> ::= "urn:" <NID> ":"<NSS>
```

8.7.3.2 Where <NID> is the namespace identifier and <NSS> is the namespace specific string. The <NID> syntax is limited to upper or lower characters followed by upper or lower characters including a hyphen "-".

```
<NID>   ::= <Letter> [<Letter-Number-Hyphen>]*
<Letter> ::= <upper> | <lower>
<Letter-Number-Hyphen> ::= <upper> | <lower> | <number> | "-"
```

8.7.3.3 The namespace identifier is case insensitive thus "ABC" and "abc" refer to the same namespace. All SEMI defined namespaces will use <NID> ::= "semi-org".

8.7.3.4 The <NSS> syntax is limited to upper or lower case characters followed by upper lower characters including a hyphen "-" . The <NSS> is composed of more than one field and each field is separated by a period as shown below. No other characters than the ones shown are allowed and should be considered reserved.

```
<NSS> ::= <XML Domain>"."<SEMI Domain>"."<Version>"."<Name>
<XML domain>  ::= <Letter> [ Letter-Number-Hyphen]
<SEMI Domain> ::= <Letter> [ Letter-Number-Hyphen]
<Version>     ::= <Letter> [ Letter-Number-Hyphen]
<Name>        ::= <Letter> [ Letter-Number-Hyphen]
```

8.7.3.5 *<XML Domain>* — A XML domain needs to be included to explain whether this is a schema (xsd) or a web service (ws) by using the following format: <XML domain> = "xsd" for schema or "ws" for web service.

8.7.3.6 *<SEMI Domain> or <Company Domain>* — Include in the SEMI domain identifier the standard number as assigned by SEMI. For example, Specification for Carrier Management = "E087". In those cases where the standard has not been approved but a formal document number has been assigned, use this number instead. For example, Equipment Diagnostic Interface = "Doc3563". When the definition is from a third party use the defined company domain. For example, Etch equipment = "EtchTool" or "Model-3731-134".

8.7.3.7 *<Version>* — Properly identifying the revision level of the schema and it is required when there is an approved version of the standard the namespace definition should include the date that corresponds to the standard release date or when it was approved. Dates are represented as month (mm) and year (yy) and use the "Vmmyy" format. This field is not required when SEMI standards are non-existent. For third party applications it is recommended that the "Vx-x" form be used. For example, 2001 Feb = "V0201" or for Version 2.1 = "V2-1"

8.7.3.8 *<Name>* — This label corresponds to the actual name of the document to which the namespace belongs. For metadata definitions include the descriptive name of the schema using upper camel case. Ex. CarrierManagement, EquipmentDiagnosticAcquisition, EquipmentMetadata, etc. For web services, the <Name> includes the service or interface role and whether it corresponds to the "binding" or the "portType" definition. (See UDDI Version 3 from OASIS for WSDL structure).

Examples of valid representations of namespaces are:

"urn:semi-org:xsd.E087.V0702.CarrierManagement"
"urn:semi-org:xsd.Doc3509.V12.DataCollectionManagement"
"urn:supplierXX-com:EtchTool.V07.Model123"
"urn:semi-org:ws.E132-1.V0305.SessionManager-binding"
"urn:semi-org:ws.E125-1.V1104.MetadataManager-portType"

Examples of invalid representation of namespaces are:

"urn:semi.org:xsd.Doc3509.V12.DataCollectionManagement" (Invalid use of period in <SEMI Domain>)
"urn:semi-org:xsd.E087.V7.22.CarrierManagement" (Invalid use of period in <Version>)

8.7.4 *Multiple Default Namespaces*

8.7.4.1 When more than one namespace is used in an XML document or schema, a prefix shall be defined for each of the namespaces that uniquely identifies the elements that belong to each of the namespaces. This prefix is used to specify that a local element or attribute name belongs to a particular namespace. These prefixes are used to override attribute values and must be documented by the supplier of the application. The following rules apply to prefixes:

Rule 1: Prefixes should be defined in all uppercase or all lower case letters but not both.

Ex:
```
<Definition xmlns: CMS="urn:semi-org:xsd.E087.V0201.CarrierManagement">
        <CMS:Data> abcd </CMS:Data>
        <CMS:Data> xyze </CMS:Data>
</Definition>
```

Rule 2: Do not use the same prefix for more than one XML namespace. Prefixes defined with the same characters and in the same order are considered the same even when defined in different letter case.

Ex. CMS and cms are equivalent.

8.7.5 *Target Namespace*

8.7.5.1 When more than one namespace is used in schema definition a target namespace should be defined using the “targetNamespace” format definition. The target namespace definition should include the elementFormDefault = “qualified” such that the elements defined within the target namespace are all qualified (in the namespace).

For example:

```
<schema targetNamespace="urn:semi-org:xsd.E125.V0303.Metadata"
        xmlns:CEM = "urn:semi-org:xsd.E120.V0303.CommonEquipmentModel"
        xmlns:CMS = "urn:semi-org:xsd.E087.V0304.CarrierManagement"
        elementFormDefault="qualified">
</schema>
```

For web services:

```
<wsdl:definitions name="ESD" targetNamespace="urn:semi-org:ws.E125-1.V0305.Metadata-portType"
        xmlns="urn:semi-org:ws.E125-1.V0305.Metadata-portType"
        xmlns:auth="urn:semi-org:xsd.E132-1.V0305.auth"
        xmlns:esd="urn:semi-org:xsd.E125-1.V0305.esd"
        xmlns:eqmport="urn:semi-org:ws.E125-1.V0305.Metadata-portType"
        xmlns:wsdl="http://schemas.xmlsoap.org/wsdl/"
        xmlns:xsd="http://www.w3.org/2001/XMLSchema">
</wsdl:definitions>
```

8.8 *Character Sets*

8.8.1 Vocabularies (element and attribute names) should be restricted to the ASCII 7-bit character set (ISO-646) in addition to restrictions imposed by XML and XML Schema on element and attribute names as defined in this document.

8.8.2 XML Schema and instance documents should use the UTF-8 (ISO-10646) encoding; all other documents should use the UTF-8 encoding (other encoding types may be used for documents whose intended use is restricted to locales where the restriction to UTF-8 is inappropriate).

8.8.3 UTF-8 supports the ASCII 7-bit character set and the Unicode character set and is the current default encoding for SOAP based documents.

8.9 *Extensions to Enumeration Lists*

8.9.1 As a general guideline fixed enumeration lists should not to be changed with additional values. Extendable enumeration lists, on the other hand, are defined such that they allow the inclusion of supplier or application specific choices.

8.9.2 *Non-extendable Enumeration Lists*

8.9.2.1 When an element or attribute has a fixed list of values it can be assumed that it is necessary to restrict the list. The reasons for restriction to a fixed list are:

- The implementation has a known set of cases to consider.
- Variations in spelling, abbreviating, and case use need to be eliminated.

For example, an answer to a question may be either “yes” or “no”. A depth may be either “HI” or “LOW”. A unit of measure may be one of a long, but finite list of ISO abbreviations.

8.9.2.2 For this case implementation is straight forward. An enumerated list is the choice and the list is not extendable. The above construct should only be used where it is clear that the list should be tightly controlled and only extended as a new version. If a user needs to extend the list they should copy this construct and build their own schema with the simpleType that contains the new set of values.

- When the usage case indicates that the enumerated list should be fixed then a simpleType using enumeration of values should be used. The elements and attributes that use these values should be local. Extension (or further restrictions) on the set of values should be done by SEMI with a change in version.

```
<xsd:simpleType name="pressureType" final="#all">   (block="#all" may also be used)
       <xsd:restriction base="xsd:string">
              <xsd:enumeration value="HI"/>
              <xsd:enumeration value="LOW"/>
       </xsd:restriction>
</xsd:simpleType>
```

8.9.3 *Extendable Fixed Enumerated Lists*

8.9.3.1 The term ‘extendable fixed’ is contradictory. However, there is a common usage case suggested for this situation. This case is where the list is fixed but allows ‘other’ or ‘unknown’ to be specified. In this case, there is no semantic meaning to the value of ‘other’ or ‘unknown’ except that it states that the given list has not appropriate values.

In this example a list of process control may include {R2R, FDC, SPC, other}. The instance writer may not want to use the listed options, and therefore can choose ‘other’ or ‘unknown’ as the value.

8.9.3.2 In this case the semantic meaning of the two values is different. ‘Other’ generally means that there is an appropriate value but it is not in the list of values that are given. ‘Unknown’ means that the instance writer does not know the value. The example below illustrates a list where the enumeration “other” defined.

```
<xs:simpleType name="ProcessType1" block="#all">
       <xs:restriction base="xs:string">
              <xs:enumeration value="R2R"/>
              <xs:enumeration value="FDC"/>
              <xs:enumeration value="SPC"/>
              <xs:enumeration value="OTHER"/>
       </xs:restriction>
</xs:simpleType>
```

- When a fixed list may not be inclusive of all possible values, include the value, ‘other.’
- When a fixed list is part of a mandatory element or attribute, consider including the value ‘unknown.’

8.9.4 *General Extendable Enumerated Lists*

8.9.4.1 In the previous section the value ‘other’ is described to be an extension policy. While this allows the user to say that there is a value other than those in the enumerated list that is appropriate, it does not define ahead of time what the value is. This is done by introducing a choice value: ‘Other: xxx’. This method allows the user to include more meaningful information that qualifies the term ‘other’.

8.9.4.2 An example where value is defined which has the pattern ‘Other: \S\S*’, with \S\S* representing a string is shown below:

```
<xs:simpleType name="otherNameType">
       <xs:restriction base="xs:string">
              <xs:pattern value="Other:\S\S*"/>
       </xs:restriction>
</xs:simpleType>
```

8.9.4.3 To make this a union with the simple type whose result is the enumerated list the example for processType illustrated above can be extended by

```
<xs:simpleType name="ProcessType2">
        <xs:restriction base="xs:string">
                <xs:enumeration value="R2R"/>
                <xs:enumeration value="FDC"/>
                <xs:enumeration value="SPC"/>
        </xs:restriction>
</xs:simpleType>
<xs:simpleType name="extProcessType">
        <xs:union memberTypes="ProcessType2 otherNameType"/>
</xs:simpleType>
```

8.9.4.4 This method allows the writer to use ‘R2R’, ‘FDC’, ‘SPC’ or ‘Other: XXX’. While it does not define the meaning of the ‘XXX’ it does allow an extension of the values at XML write time. When meanings of the extended values has been agreed upon this offers a method for incorporating these extended values without requiring schema changes..

```
<?xml version="1.0" encoding="UTF-8"?>
    <root xmlns:xsi=http://www.w3.org/2001/XMLSchema-instance
    xsi:NamespaceSchemaLocation="C:\standards\XML\E121\ExtensionSample.xsd">
      <xyz>FDC</xyz>
      <abc>Other:WTW</abc>
    </root>
```

8.9.4.5 Another advantage using this approach is that the ‘Other: WTW’ portion offers a flag to the reading application. If the reading application recognizes this string then it is aware that an extension is present and is able to take appropriate action. If this pattern is used consistently in SEMI it will become a standard pattern for the reader. Since this is a simple type it can be used as either an element value or an attribute value.

8.10 *Schema File Naming Convention*

8.10.1 Three schema files have been identified for at least each interface that a standard defines. In the case of a client interface services the three files are: DataTypes.xsd, Client-Binding.wsdl and Client-PortTypes.wsdl. Those standards that define services for multiple clients as well as for the equipment, several schema files will be defined.

8.10.2 Each defined standard that requires multiple interfaces to communicate shall provide a set of properly named schema files depending whether those services are intended for the equipment or for a particular client. Figure 1 shows an example of these schema files.

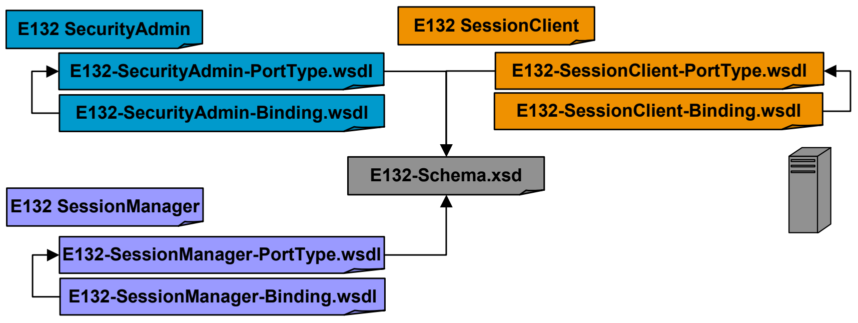
E132 SecurityAdmin
E132-SecurityAdmin-PortType.wsdl
E132-SecurityAdmin-Binding.wsdl
E132 SessionClient
E132-SessionClient-PortType.wsdl
E132-SessionClient-Binding.wsdl
E132-Schema.xsd
E132 SessionManager
E132-SessionManager-PortType.wsdl
E132-SessionManager-Binding.wsdl

Figure 1
Example of Schema File Organization

8.10.3 The following convention will be used with SEMI standards:

<Standard Name>-<Version>[-<Service Role>]-<Schema Classification>.<Schema Type>

Where:

<Standard Name>::= Short name of the standard or SEMI assigned “E” number (Ex. E125-1 or E132-1 or E128 or CommonComponent)
<Version> ::= V0305 (Release or publication date of the standard)
<Service Role> ::= Only applies to web services or wsdl files . (Ex. SessionClient, SessionManager, SecurityAdmin)
<Schema Classification> ::= “Schema”, “Binding” or ”PortType”.
<Schema Type> ::= xsd or wsdl

For data type definition schema files the following are acceptable names:

Unit-V0305-Schema.xsd
E125-V1104-Schema.xsd
CommonComponents-V0705-Schema.xsd

For Web Service files the following are acceptable names:

E125-1-V0305-MetadataClient-Binding.wsdl
E132-1-V1104-SessionClient-PortType.wsdl
E132-1-V1104-SessionManager-PortType.wsdl

RELATED INFORMATION 1

NOTICE: This related information is not an official part of SEMI E121 and was derived from the work of the North American Information and Control Committee. This related information was approved for publication by full letter ballot on November 4, 2004.

R1-1 Overview of XML Resources, Organizations and Standards

R1-1.1 There are an extensive number of resources in the area of XML. Most of the information is readily available from organizations such as OMG, W3C, RosettaNet, ebXML, OASIS, Sun Micro Systems, IBM and others through their web sites. There has been a lot of work standardizing dictionaries and vocabularies for several industries and these documents can be used until further work is generated in this industry. Books are also widely available from several editorial companies like O'Reily, Wrox, SAMS, Addison Wesley and other to mention some. Titles vary from publisher to publisher but the information is in general the same. Next, is a list of these organizations and their role in the development of standards for XML.

R1-1.2 *OMG*

R1-1.2.1 The Object Management Group (OMG) is an open membership, not-for-profit consortium that produces and maintains computer industry specifications for interoperable enterprise applications. The membership roster, comprises of about 800 scientists from the industry, includes virtually every large company in the computer industry, and hundreds of smaller ones. Their flagship specification is the multi-platform Model Driven Architecture (MDA), recently underway but already well known in the industry. It is based on the modeling specifications the MOF, the UML, XMI, and CWM. OMG's own middleware platform is CORBA, which includes the Interface Definition Language OMG IDL, and protocol IIOP. The Object Management Architecture (OMA) defines standard services that will carry over into MDA work shortly. OMG Task Forces standardize Domain Facilities in industries such as healthcare, manufacturing, telecommunications, and others. This organization is of interest because of the work made in the areas of UML, XMI and MOF. Link information: www.OMG.org

R1-1.3 *W3C*

R1-1.3.1 The World Wide Web Consortium was created in October 1994 to lead the World Wide Web to its full potential by developing common protocols that promote its evolution and ensure its interoperability. W3C has more than 500 Member organizations from around the world and has earned international recognition for its contributions to the growth of the Web. By promoting interoperability and encouraging an open forum for discussion, W3C commits to leading the technical evolution of the Web. In just over seven years, W3C has developed more than 35 technical specifications for the Web's infrastructure. Some examples of those technical specifications are: The XML 1.0 Recommendation (published in February 1998). This was the first step towards the next generation Web, allowing each community to design languages that suit their particular needs and integrate them harmoniously into a general infrastructure based on XML. Since XML 1.0, a number of Recommendations have added to the XML infrastructure: XML Namespaces was published in January 1999, Associating Style Sheets with XML documents were published in June 1999, and XSLT, for XML transformations, was published in November 1999. Web services and SOAP (Simple Object Application Protocol) recommendations are being voted and will be published in a near date. Link information: www.w3.org

R1-1.4 *ebXML*

R1-1.4.1 ebXML (Electronic Business using eXtensible Markup Language), sponsored by UN/CEFACT and OASIS, is a modular suite of specifications that enables enterprises of any size and in any geographical location to conduct business over the Internet. Using ebXML, companies now have a standard method to exchange business messages, conduct trading relationships, communicate data in common terms and define and register business processes. Technical work on ebXML moves forward as a coordinated activity between members of UN/CEFACT and OASIS. Some of the standards driven by this consortia are: ebXML Messaging Services, Registries and Repositories, Collaborative Protocol Profile and Implementation, Interoperability and Conformance work is conducted within the OASIS technical process, because of the Consortium's expertise in XML standards development. ebXML Core Components and Business Process Models is advanced within UN/CEFACT, because of its vast experience in EDI standards. Link information: www.ebXML.org

R1-1.5 *RossetaNet*

R1-1.5.1 A consortium of more than 400 of the world's leading Electronic Components (EC), Information Technology (IT), Semiconductor Manufacturing (SM) and Solution Provider (SP) companies, RosettaNet is a self-funded, non-profit organization dedicated to creating, implementing and promoting open e-business standards. These standards form a common e-business language, aligning processes between trading partners on a global basis. RosettaNet is named after the Rosetta stone, which, carved with the same message in three languages, led to the understanding of hieroglyphics. RosettaNet, like the Stone, is breaking language barriers and making history. RosettaNet drives collaborative development and rapid deployment of Internet-based business standards, creating a common language and open e-business processes that provide measurable benefits and are vital to the evolution of the global, high technology trading network. Developed with the collaboration and expertise of leading high-tech companies, RosettaNet standards offer a robust nonproprietary solution, encompassing data dictionaries, implementation framework, and business message schemas and process specifications, for e-business standardization. RosettaNet standards include the RosettaNet Business Dictionary, RosettaNet Technical Dictionary, RosettaNet Implementation Framework (RNIF) and RosettaNet Partner Interface Processes® (PIP®). Link information: www.rosetta.org

R1-1.6 *UN/CEFACT*

R1-1.6.1 UN/CEFACT is the United Nations body whose mandate covers worldwide policy and technical development in the area of trade facilitation and electronic business. Headquartered in Geneva, it has developed and promoted many tools for the facilitation of global business processes including UN/EDIFACT, the international EDI standard. Its current work "programme" includes such topics as "Simpl-edi" and "Object Oriented edi" and it strongly supports the development and implementation of open interoperable, global standards and specifications for electronic business. Link information: www.unece.org/cefact

R1-1.7 *OASIS*

R1-1.7.1 OASIS is the international, not-for-profit consortium that advances electronic business by promoting open, collaborative development of interoperability specifications. The XML.ORG Registry provides as an open community clearinghouse for distributing and locating XML application schemas, vocabularies and related documents. OASIS serves as the home for industry groups and organizations interested in developing XML specifications. OASIS is a not-for-profit, global consortium that drives the development, convergence and adoption of e-business standards. OASIS produces worldwide standards for security, Web services, XML conformance, business transactions, electronic publishing, topic maps and interoperability within and between market places. OASIS has more than 400 corporate and individual members in 100 countries around the world. OASIS and the United Nations jointly sponsor ebXML, a global framework for e-business data exchange. OASIS operates XML.org, a community clearinghouse for XML application schemas, vocabularies and related documents. OASIS hosts The XML Cover Pages, an online reference collection for interoperable markup language standards. The OASIS Network includes CGM Open and LegalXML. Link information: http://www.oasis-open.org

R1-1.8 *NIST*

R1-1.8.1 Founded in 1901, NIST is a non-regulatory federal agency within the U.S. Commerce Department's Technology Administration. NIST's mission is to develop and promote measurements, standards, and technology to enhance productivity, facilitate trade, and improve the quality of life. NIST carries out its mission in four cooperative programs: NIST Laboratories, conducting research that advances the nation's technology infrastructure and is needed by U.S. industry to continually improve products and services; the Baldrige National Quality Program, which promotes performance excellence among U.S. manufacturers, service companies, educational institutions, and health care providers; conducts outreach programs and manages the annual Malcolm Baldrige National Quality Award which recognizes performance excellence and quality achievement; the Manufacturing Extension Partnership, a nationwide network of local centers offering technical and business assistance to smaller manufacturers; and the Advanced Technology Program, which accelerates the development of innovative technologies for broad national benefit by co-funding R&D partnerships with the private sector. One program in particular is SIMA. The SIMA Program seeks to provide U.S. manufacturers with capabilities enabling contextually meaningful data to be shared among business activities such that reliable information is accessible when and where it is needed. Link information: www.NIST.gov

R1-1.9 *ISO*

R1-1.9.1 The International Organization for Standardization (ISO) is a worldwide federation of national standards bodies from some 140 countries, one from each country. ISO is a non-governmental organization established in 1947. The mission of ISO is to promote the development of standardization and related activities in the world with a view to facilitating the international exchange of goods and services, and to developing cooperation in the spheres of intellectual, scientific, technological and economic activity. ISO's work results in international agreements that are published as International Standards. This guide may use some of the recommendations defined in ISO 11179 Document that defines a framework for the specification and standardization of data elements. Link information: www.ISO.org.

R1-2 Glossary of Terms in XML

R1-2.1 *Attribute* — A qualifier on an XML tag that provides additional information. For example, in the tag <slide: title="My Slide"/>, title is an attribute, and My Slide is its value.

R1-2.2 *Binding* — Construction of the code needed to process a well-defined bit of XML data.

R1-2.3 *Comment* — Text in an XML document that is ignored, unless the parser is specifically told to recognize it. A comment is enclosed in a comment tag, like this: <!-- This is a comment --> .

R1-2.4 *Content* — The part of an XML document that occurs after the prolog, including the root element and everything it contains.

R1-2.5 *CDATA* — A predefined XML tag for "Character DATA" that says "don't interpret these characters", as opposed to "Parsed Character Data" (PCDATA), in which the normal rules of XML syntax apply (for example, angle brackets demarcate XML tags, tags define XML elements, etc.). CDATA sections are typically used to show examples of XML syntax. Like this:

```
<![CDATA[ <slide>..A sample slide..</slide> ]]>
```

which displays as:

```
<slide>..A sample slide.. </slide>
```

R1-2.6 *Data* — The contents of an element generally used when the element does not contain any sub elements. When it does, the more general term content is generally used. When the only text in an XML structure is contained in simple elements, and elements that have sub elements have little or no data mixed in, then that structure is often thought of as XML "data", as opposed to an XML document.

R1-2.7 *Declaration* — The very first thing in an XML document, which declares it as XML. The minimal declaration is <?xml version="1.0"?>. The declaration is part of the document prolog.

R1-2.8 *Document* — In general, an XML structure in which one or more elements contain text intermixed with sub elements. See also: data.

R1-2.9 *DOM* — Document Object Model. A tree of objects with interfaces for traversing the tree and writing an XML version of it, as defined by the W3C specification.

R1-2.10 *Element* — A unit of XML data, delimited by tags. An XML element can enclose other elements. For example, in the XML structure, "<slideshow><slide>..</slide><slide>..</slide></slideshow>", the <slideshow> element contains two <slide> elements.

R1-2.11 *Entity* — A distinct, individual item that can be included in an XML document by referencing it. Such an entity reference can name an entity as small as a character (for example, "&lt" , which references the less-than symbol, or left-angle bracket (<). An entity reference can also reference an entire document, or external entity, or a collection of element definitions (a parameter entity).

R1-2.12 *Entity Reference* — A reference to an entity that is substituted for the reference when the XML document is parsed. It may reference a predefined entity like &lt; or it may reference one that is defined in the schema. In the XML data, the reference could be to an entity that is defined in the local subset of the schema or to an external XML file (an external entity). The schema can also carve out a segment of schema specifications and give it a name so that it can be reused (included) at multiple points in the schema by defining a parameter entity.

R1-2.13 *Error* — A SAX parsing error is generally a validation error—in other words, it occurs when an XML document is not valid, although it can also occur if the declaration specifies an XML version that the parser cannot handle. See also: fatal error, warning.

R1-2.14 *External Entity* — An entity that exists as an external XML file, which is included in the XML document using an entity reference.

R1-2.15 *External Subset* — That part of the schema that is defined by references to external .dtd or .xsd files.

R1-2.16 *Fatal Error* — A fatal error occurs in the SAX parser when a document is not well formed, or otherwise cannot be processed. See also: error, warning.

R1-2.17 *General Entity* — An entity that is referenced as part of an XML document's content, as distinct from a parameter entity, which is referenced in the schema. A general entity can be a parsed entity or an unparsed entity.

R1-2.18 *HTML* — Hypertext Markup Language. The language of the Web. A system where every document has a globally unique location, and documents can link to one another.

R1-2.19 *Local Subset* — That part of the schema that is defined within the current XML file.

R1-2.20 *Namespace* — A standard that lets you specify a unique label to the set of element names defined by a schema. A document using that schema can be included in any other document without having a conflict between element names. The elements defined in your schema are then uniquely identified so that, for example, the parser can tell when an element called <name> should be interpreted according to your schema, rather than using the definition for an element called "name" in a different schema.

R1-2.21 *Normalization* — The process of removing redundancy by modularizing, as with subroutines, and have removing superfluous differences by reducing them to a common denominator. For example, reducing them to a single NL normalizes line endings from different systems, and multiple white space characters are normalized to one space.

R1-2.22 *Notation* — A mechanism for defining a data format for a non-XML document referenced as an unparsed entity. This is a holdover from SGML that creaks a bit. The newer standard is to use MIME data types and namespaces to prevent naming conflicts.

R1-2.23 *OASIS* — Organization for the Advancement of Structured Information Standards. Their home site is http://www.oasis-open.org/. The schema repository they sponsor is at http://www.XML.org.

R1-2.24 *Parsed Entity* — A general entity which contains XML, and which is therefore parsed when inserted into the XML document, as opposed to an unparsed entity.

R1-2.25 *Parser* — A module that reads in XML data from an input source and breaks it up into chunks so that your program knows when it is working with a tag, an attribute, or element data. A no validating parser ensures that the XML data is well formed, but does not verify that it is valid. See also: validating parser.

R1-2.26 *Processing Instruction* — Information contained in an XML structure that is intended to be interpreted by a specific application.

R1-2.27 *Prolog* — The part of an XML document that precedes the XML data. The prolog includes the declaration and an optional schema.

R1-2.28 *Reference* — See entity reference.

R1-2.29 *RDF* — Resource Description Framework. A standard for defining the kind of data that an XML file contains. Such information could help ensure semantic integrity, for example by helping to make sure that a date is treated as a date, rather than simply as text.

R1-2.30 *RDF Schema* — A standard for specifying consistency rules (for example, price must be greater than zero, discount must be less than 15%) that apply to the specifications contained in an RDF.

R1-2.31 *Root* — The outermost element in an XML document. The element that contains all other elements.

R1-2.32 *SAX* — “Simple API for XML”. An event-driven interface in which the parser invokes one of several methods supplied by the caller when a “parsing event” occurs. “Events” include recognizing an XML tag, finding an error, encountering a reference to an external entity, or processing a schema specification.

R1-2.33 *Schema* — A database-inspired method for specifying constraints on XML documents using an XML-based language. Schemas address deficiencies in DTDs, such as the inability to put constraints on the kinds of data that can occur in a particular field (for example, all numeric). Since schemas are founded on XML, they are hierarchical, so it is easier to create an unambiguous specification, and possible to determine the scope over which a comment is meant to apply.

R1-2.34 *SGML* — Standard Generalized Markup Language. The parent of both HTML and XML. However, while HTML shares SGML’s propensity for embedding presentation information in the markup, XML is a standard that allows information content to be totally separated from the mechanisms for rendering/displaying that content.

R1-2.35 *Tag* — A piece of text that describes a unit of data, or element, in XML. The tag is distinguishable as markup, as opposed to data, because it is surrounded by angle brackets (< and >). For example, the element <name>My Name</name> has the start tag <name>, the end tag </name>, which enclose the data “My Name”. To treat such markup syntax as data, you use an entity reference or a CDATA section.

R1-2.36 *Unicode* — A standard defined by the Unicode Consortium that uses a 16-bit “code page” which maps digits to characters in languages around the world. Because 16 bits covers 32,768 codes, Unicode is large enough to include all the world’s languages, with the exception of ideographic languages that have a different character for every concept, like Chinese. For more info, see http://www.unicode.org/.

R1-2.37 *Unparsed Entity* — A general entity that contains something other than XML. By its nature, then, an unparsed entity contains binary data.

R1-2.38 *URI* — A “Universal Resource Identifier”. A URI is either a URL or a URN. (URLs and URNs are concrete entities that actually exist. A “URI” is an abstract superclass -- it’s a name we can use when we know we are dealing with either an URL or an URN, and we don’t care which.

R1-2.39 *URL* — Universal Resource Locator. A pointer to a specific location (address) on the Web that is unique in the entire world. The first part of the URL defines the type of address. For example, http:/ identifies a Web location. The ftp:/ prefix identifies a downloadable file. Other prefixes include file:/ (a file on the local disk system) and mailto:/ (an email address).

R1-2.40 *URN* — Universal Resource Name. A unique identifier that identifies an entity, but doesn’t tell where it is located. That lets the system look it up to see if a local copy exists before going out to find it on the Web. It also allows the web location to change, while still allowing the object to be found.

R1-2.41 *Valid* — A valid XML document, in addition to being well formed, conforms to all the constraints imposed by a schema. In other words, it does not contain any tags that are not permitted by the schemas, and the order of the tags conforms to the or schema’s specifications.

R1-2.42 *Validating Parser* — A validating parser is a parser that ensures that an XML document is valid, as well as well formed. See also: parser.

R1-2.43 *w3c* — The World Wide Web Consortium. The international body that governs Internet standards.

R1-2.44 *Warning* — A SAX parser warning is generated when the document’s schema contains duplicate definitions, and similar situations that are not necessarily an error, but which the document author might like to know about, since they could be. See also: fatal error, error.

R1-2.45 *Well-formed* — A well-formed XML document is syntactically correct. It does not have any angle brackets that are not part of tags. (The entity references &lt; and &gt; are used to embed angle brackets in an XML document.) In addition, all tags have an ending tag or are themselves self-ending (<slide>..</slide> or <slide/>). In addition, in a well-formed document, all tags are fully nested. They never overlap, so this arrangement would produce an error: <slide><image>..</slide></image>. Knowing that a document is well formed makes it possible to process it. A well-formed document may not be valid however. To determine that, you need a validating parser and a schema.

R1-2.46 *XHTML* — An XML look alike for HTML defined by one of several XHTML schemas. To use XHTML for *everything* would of course defeat the purpose of XML, since the idea of XML is to identify information content, not just tell how to display it. XHTML makes the conversion from HTML to XML, though. You can also reference it in a schema, which allows you to say, for example, that the text in an element can contain <em> and <b> tags, rather than being limited to plain text.

R1-2.47 *XLink* — The part of the XLL specification that is concerned with specifying links between documents.

R1-2.48 *XLL* — The XML Link Language specification, consisting of XLink and XPointer.

R1-2.49 *XML* — Extensible Markup Language, which allows you to define the tags (markup) that you need to identify the data and text in XML documents.

R1-2.50 *XML Schema* — The w3c schema specification for XML documents.

R1-2.51 *XPath* — See XSL.

R1-2.52 *Xpointer* — The part of the XLL specification that is concerned with identifying sections of documents so that they can referenced in links or included in other documents.

R1-2.53 *XSL* — Extensible Stylesheet Language. An important standard that achieves several goals. XSL lets you:

a) Specify an addressing mechanism, so you can identify the parts of an XML file that a transformation applies to (**XPath**).

b) Specify tag conversions, so you convert XML data into different formats. (**XSLT**).

c) Specify display characteristics, such page sizes, margins, and font heights and widths, as well as the *flow objects* on each page. Information fills in one area of a page and then automatically flows to the next object when that area fills up. That allows you to wrap text around pictures, for example, or to continue a newsletter article on a different page. (**XML-FO**)

R1-2.54 *XSL-FO* — See XSL.

R1-2.55 *XSLT* — See XSL.

SEMI E122-0703
STANDARD FOR TESTER EQUIPMENT SPECIFIC EQUIPMENT MODEL (TSEM)

This standard was technically approved by the Global Information and Control Committee and is the direct responsibility of the North American Information and Control Committee. Current edition approved by the North American Regional Standards Committee on November 22, 2002. Initially available at www.semi.org January 2003; to be published March 2003.

NOTICE: The designation of SEMI E122 was updated during the 0703 publishing cycle to reflect the addition of SEMI E122.1.

NOTICE: This standard replaces SEMI E30.3, which has been removed from publication as of the March 2003 (0303) publication cycle.

1 Purpose

1.1 This document establishes a Specific Equipment Model for testing equipment (TSEM). The TSEM consists of equipment characteristics and behaviors that apply to this class of equipment. These characteristics and behaviors are required to be implemented. The intent of this document is to facilitate the integration of testing equipment into an automated semiconductor factory. This document accomplishes this by defining an operational model for testing equipment as viewed by a factory automation controller. This definition provides a standard host interface and equipment operational behavior.

2 Scope

2.1 The document defines the view of the equipment through the host communications link but does not define the internal operation of the equipment. It includes a specific processing state model as the basis for the behavior of all equipment of this class.

2.2 This document expands testing equipment requirements and capabilities in the areas of the processing state model, collection events, alarm documentation, remote commands, variable items, and process program management.

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 *Communications*

3.1.1 It is required that any TSEM-compliant equipment follow the Communications State Model in SEMI E30. In addition TSEM-compliant equipment shall support the High-Speed Messaging Service (SEMI E37) Communication Standard sending messages over TCP/IP to maximize the amount of data available for monitoring from this class of equipment. This specification deals only with the behavior of the tester in communicating with the host application

3.2 *Virtual Tester*

3.2.1 Section 7 of this document describes the concept of the Virtual Tester. The requirements described in this document must be fully implemented on each virtual tester, independent of any other virtual tester. For the purposes of this document, it can be assumed that the term "tester" refers to the virtual tester as defined in Section 7. Any references to the physical tester will be explicitly noted.

4 Referenced Standards

4.1 *SEMI Standards*

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

SEMI E37 — High-Speed SECS Message Services (HSMS) Generic Services

SEMI E37.1 — High-Speed SECS Message Services Single Selected-Session Mode (HSMS-SS)

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

5 Terminology

5.1 *Definitions*

5.1.1 *calibration fixture* — any electromechanical fixture required to perform system calibration. The calibration fixture may consist of multiple components with different part and serial numbers.

5.1.2 *class* — classes represent the most coarse view of the test results. At a minimum, there should be two classes defined for each process program: one class representing good units and another class representing failed units.

5.1.3 *datalog* — collection of results of individual test measurements gathered during the execution of the test program.

5.1.4 *diagnostic fixture* — any electromechanical fixture required to perform system diagnostics. The diagnostic fixture may consist of multiple components with different parts and serial numbers.

5.1.5 *execution area* — the area from which a current copy of the process program instructions is executed.

5.1.6 *hard-bin* — hard-bins represent the typical view of the test results. Within a process program, each hard-bin is associated with a single class. Generally, multiple hard-bins are associated with a particular class.

5.1.7 *multi-site testing* — testing of multiple units with one execution of the test program. Each unit has it own test results.

5.1.8 *soft-bin* — soft-bins represent the most detailed view of the test results. Within a process program, each soft-bin is associated with a single hard-bin. Generally, multiple soft-bins are associated with a particular hard-bin.

5.1.9 *station controller (SC)* — the station controller consists of software that coordinates the actions of the test system and the unit handling equipment (wafer prober, package handler, etc.). It may reside on the test system computer or some other computer. One station controller may be in charge of one or more virtual testers.

5.1.10 *system calibration* — test system process required to bring the test system into compliance with the test system manufacturer's system specifications.

5.1.11 *test-board* — the electromechanical interface necessary to enable temporary electrical contact between the unit to be tested and the tester resource. The test-board may consist of multiple components.

5.1.12 *test-head* — a resource of the tester. The test-head is the electromechanical interface between the unit and the tester.

5.1.13 *test-site* — a location on a test-board where one unit at a time is positioned for testing.

5.1.14 *tester executive* — the tester software which controls test program execution.

5.1.15 *testing equipment* — an equipment class generally consisting of integrated mechanisms and controls for performing electrical tests of packaged devices and/or wafer die (units) during the manufacturing process.

5.1.16 *unit* — the functional integrated circuit (or chip) that is to be electrically tested.

5.1.17 *virtual tester* — the virtual tester is a logical concept which describes the dynamic allocation of a portion of the physical hardware available in a test system to a test program that uses those resources to test a unit or set of units.

6 State Model

6.1 The purpose is to define the equipment-specific processing state model and other state models necessary to portray the expected operational states of the equipment to enable host tracking and control in place of a local operator.

6.2 *State Model Requirements*

6.2.1 The processing state model in this document is required for implementing a TSEM-compliant tester. A state model consists of a processing state model diagram, processing state definitions, and a processing state transition table. A state model represents the host's view of the virtual tester, but not necessarily the actual tester operations. All TSEM state model transitions shall be mapped sequentially into the actual equipment events that satisfy the requirements of those transitions. In certain implementations, the tester may enter a state and have already satisfied all of the conditions required by the TSEM state model for transition to another state. In this situation, the tester makes the required transition without any additional actions.

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

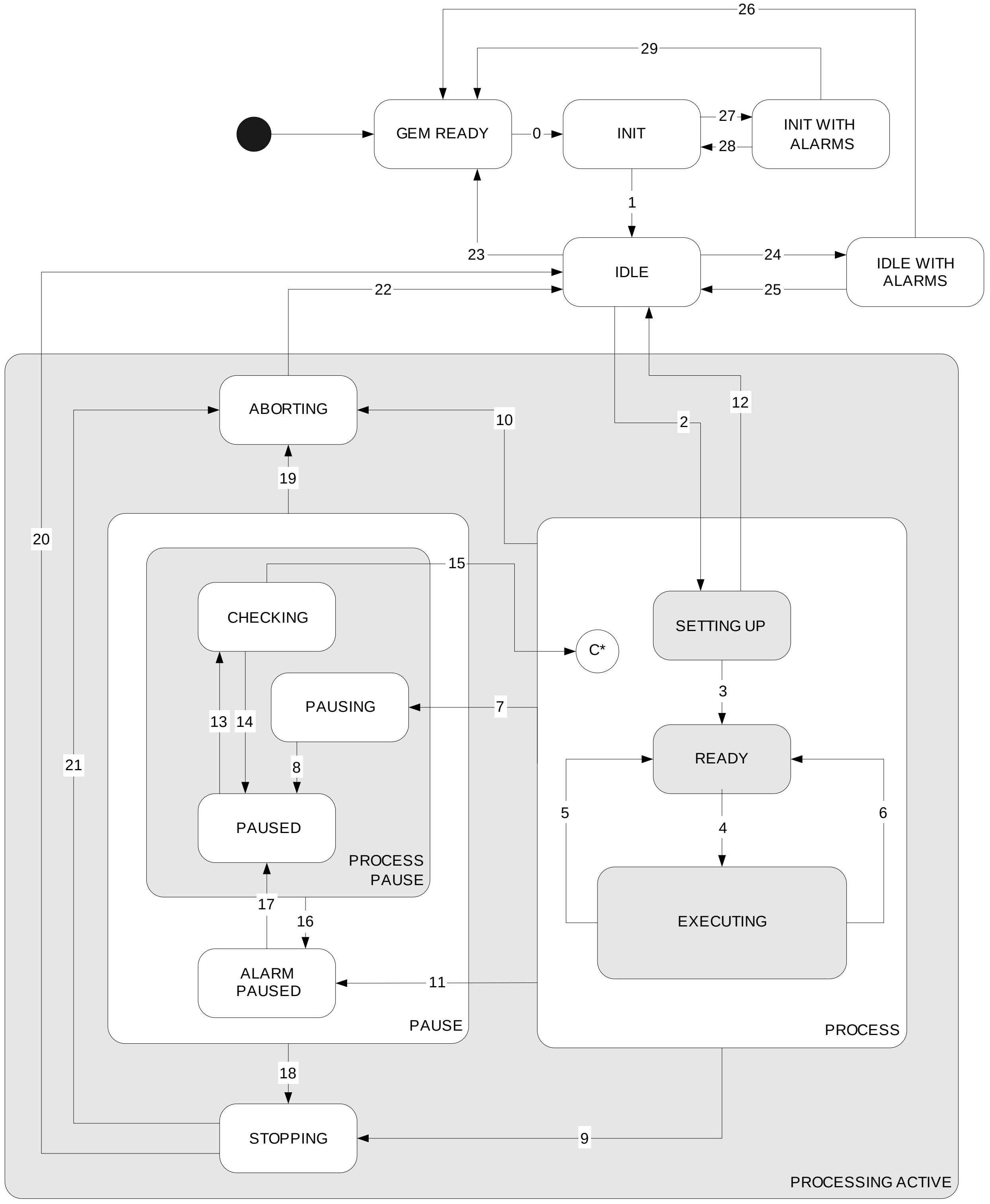

Figure 1
TSEM Processing State Model

6.3 *Description of Tester Processing States*

6.3.1 *ABORTING (PROCESSING ACTIVE Sub-State)* — The tester has received an ABORT command. All activity is immediately suspended. The tester is taking appropriate action to bring itself and material to a "safe" state where possible.

6.3.2 *ALARM PAUSED (PAUSE Sub-State)* — An alarm has occurred in the PROCESS or PROCESS PAUSE states and the tester is waiting for the alarm to be cleared.

6.3.3 *CHECKING (PROCESS PAUSE Sub-State)* — The tester verifies that updates made to the process program are valid. Process program updates result in the definition of a new process program. This is a similar procedure to that which is done in SETTING UP. If there are no process program updates, this is a pass through state. At the successful completion of verification, a transition, based on the process model condition table, is made to the process state.

6.3.4 *EXECUTING (PROCESS Sub-State*) — The tester is processing material.

6.3.5 *IDLE* — The tester executive is loaded and the tester is awaiting a command. IDLE is free of ALARMS and error conditions. If this state is entered with valid alarms (e.g. as a result of an abnormal exit from ABORTING, STOPPING, or SETTING UP) then this state is a pass through to IDLE WITH ALARMS.

6.3.6 *IDLE WITH ALARMS* — An alarm has occurred in the IDLE state and the tester is waiting for all alarms to be cleared. An operator may be required to clear alarm conditions. The system shall remain in this state until all alarm conditions are cleared, including those that require operator intervention.

6.3.7 *INIT* — Tester initialization is occurring. The tester executive is loading.

6.3.8 *INIT WITH ALARMS* — An alarm occurred during the initialization process.

6.3.9 *PAUSE (PROCESSING ACTIVE Sub-State)* — This state is the parent of those sub-states in which the process program is paused.

6.3.10 *PAUSED (PROCESS PAUSE Sub-State)* — The PROCESS state has been suspended and the tester is waiting for a command (RESUME, STOP, or ABORT). In this state, the operator may modify conditions of the current Process Program selection.

6.3.11 *PAUSING (PROCESS PAUSE Sub-State)* — The current state will be suspended at the completion of the current unit(s), if any, and the tester will be brought to a "safe" state.

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

6.3.13 *PROCESS PAUSE (PAUSE Sub-State)* — This state is the parent state of the sub-states that refer to the pausing of tester operation. The tester is free of alarm conditions in the PAUSE state.

6.3.14 *PROCESSING ACTIVE* — This state is the parent state of the sub-states that know the identity of the process program.

6.3.15 *READY (PROCESS Sub-State)* — The process program is loaded. The tester is ready to begin processing and is awaiting a START command.

6.3.16 *SETTING UP (PROCESS Sub-State)* — The selected process program is loading. The tester is satisfying conditions so that processing can begin. This includes the initialization and validation of the process program and process program-specific calibration. This may be accomplished independently by the tester or may require interaction with the operator and/or host.

6.3.17 *STOPPING (PROCESSING ACTIVE Sub-State)* — The tester has been instructed to stop processing and shall do so at the next opportunity. All necessary cleanup is completed within this state with regard to material, data, control system, etc.

6.3.18 *GEM READY* — The GEM session is running and ready to accept a START EXEC command from the host.

6.4 *TSEM Processing State Transitions Table*

Table 1 Processing State Transitions Table

#	*Current State*	*Trigger*	*New State*	*Actions*	*Comments*
0	GEM READY	The GEM session has received a START EXEC command.	INIT	If first START EXEC on this physical tester, start the tester executive.	The first tester executive to start initializes all hardware.
1	INIT	All tester initialization is complete with no alarms or error conditions.	IDLE	None.	The tester executive is running.
2	IDLE	A process program is selected.	SETTING UP	Tester-dependent.	Commit has been made to setup and a virtual tester has been defined.
3	SETTING UP	All setup activity has completed and the tester is ready to receive a START command.	READY	None.	The selected Process Program is available for execution, all required hardware for this virtual tester is allocated, and the test program is loaded.
4	READY	The tester receives a START command from a valid source.	EXECUTING	Begins testing the unit(s) at the test-site(s).	The definition of a valid source is application dependent.
5	EXECUTING	The processing of the current unit(s) has completed normally.	READY	The tester processes end of test data.	"Normal" completion of the test program execution.
6	EXECUTING	The processing of the current unit(s) has completed abnormally.	READY		"Abnormal" completion of the test program execution, etc.
7	PROCESS	The tester has received a PAUSE command from a valid source.	PAUSING	The current state is suspended at the completion of the current unit(s).	The definition of a valid source is application dependent.
8	PAUSING	The tester has completed processing the current unit(s).	PAUSED	The tester is waiting for a command (RESUME, STOP, or ABORT).	None.
9	PROCESS	The tester has received a STOP command from a valid source.	STOPPING	The tester executes the actions required to perform a STOP.	The definition of a valid source is application dependent.
10	PROCESS	The tester has received an ABORT command from a valid source.	ABORTING	The tester executes the actions required to perform an ABORT.	The definition of a valid source is application dependent.
11	PROCESS	An alarm occurs.	ALARM PAUSED	PROCESS activity is suspended and the tester is waiting for all alarms to be cleared.	
12	SETTING UP	An equipment initiated transition indicating that the process program failed to load.	IDLE	The hardware resources are released. If the process program load failure caused an alarm condition, this transition is followed immediately by a transition 24 to IDLE WITH ALARMS.	No process program is loaded.
13	PAUSED	The tester receives a RESUME command.	CHECKING	If the RESUME command contains variable process program parameters, validation of the process program parameters begins.	If there are no parameters on the RESUME command, the CHECKING state is a pass through.
14	CHECKING	Error detected in a new parameter setting.	PAUSED	The tester waits for the parameter correction by operator or host.	The definition of a valid source is application dependent.

#	*Current State*	*Trigger*	*New State*	*Actions*	*Comments*
15	CHECKING	Parameter checking completes successfully.	State based on conditional table.	The tester resumes PROCESS execution.	This is a conditional re-entry to the PROCESS state. (See Table 2.) If there are no parameters, the CHECKING state is a pass through.
16	PROCESS PAUSE	An alarm is set.	ALARM PAUSED	The tester waits for all alarms to be cleared.	None.
17	ALARM PAUSED	All alarms are cleared.	PAUSED	An operator has verified that all alarms and abort conditions have been cleared. The tester is waiting for a valid command.	None.
18	PAUSE	The tester has received a STOP command from a valid source.	STOPPING	The tester executes the actions required to perform a STOP.	The definition of a valid source is application dependent.
19	PAUSE	The tester has received an ABORT command from a valid source.	ABORTING	The tester executes the actions required to perform an ABORT.	The definition of a valid source is application dependent.
20	STOPPING	The tester completes the actions required to perform a STOP.	IDLE	The hardware resources for this virtual tester are released.	
21	STOPPING	The tester has received an ABORT command from a valid source.	ABORTING	The tester executes the actions required to perform an ABORT.	The definition of a valid source is application dependent.
22	ABORTING	The tester has completed the action required to perform an ABORT.	IDLE	The hardware resources for this virtual tester are released. Unsafe conditions have been resolved where possible. An operator has verified that all alarms and abort conditions have been cleared.	
23	IDLE	The tester receives an STOP EXEC command from a valid source.	GEM READY	The tester executive is unloaded from all testers.	The definition of a valid source is application dependent.
24	IDLE	An alarm is set.	IDLE with ALARMS	The tester waits for all alarms to be cleared.	If the ALARM already exists upon entering the IDLE state, the tester transitions immediately from IDLE to IDLE with ALARMS.
25	IDLE WITH ALARMS	All alarms on the tester have been cleared.	IDLE	An operator has verified that all alarms and abort conditions have been cleared.	The IDLE state is free of alarms.
26	IDLE WITH ALARMS	The tester executive has been stopped by an STOP EXEC command form a valid source.	GEM READY	An operator has verified that all alarms and abort conditions have been cleared. The tester executive is unloaded from all testers.	The definition of a valid source is application dependent.
27	INIT	An error occurred during the loading of the tester exec.	INIT WITH ALARMS		None.
28	INIT WITH ALARMS	Alarms are cleared.	INIT	An operator has verified that all alarms and abort conditions have been cleared.	None.

#	*Current State*	*Trigger*	*New State*	*Actions*	*Comments*
29	INIT WITH ALARMS	The tester executive has been stopped by an STOP EXEC command form a valid source.	GEM READY	An operator has verified that all alarms and abort conditions have been cleared. The tester executive is unloaded from all virtual testers.	The definition of a valid source is application dependent.

6.5 *Process Model Conditions Table*

Table 2 Process Resume Conditions

Condition	*Next State*
Checking determines that process program conditions were changed.	SETTING UP
Previous state EXECUTING, and no process program conditions were changed.	READY
Previous state READY, and no process program conditions were changed.	READY
Previous state was SETTING UP.	SETTING UP

7 Virtual Testers

7.1 *General Definitions*

7.1.1 The virtual tester is defined as a single load instance of a test program (regardless of how many test-sites, test system resources, or test-heads are associated with that instance) that is capable of operating as an independent tester in accordance with the state model shown in this document.

7.1.2 Loading another instance of the same test program, which uses separate or shared test system hardware resources, establishes another virtual tester. The multiple copies of that test program are independent of each other. In this context, program independence describes the condition that one virtual tester program does not expect or rely on the existence of another virtual tester program in order to run.

7.1.3 For a single test system with one test-head that must be dedicated to a single test program, the physical tester and virtual tester are the same and a single GEM communication interface is needed. In a test system with two test-heads, where each test-head can execute a unique process program autonomously, two virtual testers may be operating at the same time with two GEM communication sessions. If a test-head may be divided into several segments, each capable of autonomous execution of a unique program, there will be one virtual tester and one matching GEM communication session for each of those segments.

7.1.4 The number of virtual testers operating at any time depends on how the test system is configured (hardware and software) and may range from one to the maximum capability of the particular test system. The number of GEM communication sessions is equal to the number of virtual testers supported by the tester.

7.2 *GEM Communication Sessions*

7.2.1 From the GEM communication host connection, only one program and virtual tester exists and the host GEM communication connection has no knowledge or view of other virtual testers. Independence does not preclude the tester executive or test hardware from sharing resources between virtual testers, such as in the case of power supplies or shared software libraries.

7.2.2 For each virtual tester, the test system must provide a separate GEM client connection. The test system controller has responsibility to manage the multiple client connections.

7.2.3 For each virtual tester the station controller establishes a unique host connection to a single tester GEM client. The station controller has responsibility to manage the multiple host connections.

7.2.4 The test system will have a set maximum number of allowable unique GEM host/client connections. The maximum number of connections (n) and the port IDs for these GEM communication connections are constants known to the station controller and available from the test system controller after that controller is booted.

7.2.5 The maximum number of GEM communication sessions allowed by the test system is also the maximum number of virtual testers allowed by the test system, regardless of the number and allocation of tester resources.

7.2.6 Each unique virtual tester host/client connection is an independent GEM communication session. The following hierarchy relationship can be established.

7.2.7 GEM Communication session (1 ... n) accepts program load instances (1 ... n) which uses tester hardware resources (x ... y) which establishes Virtual Tester (1 ... n). This relationship does not imply any order of assignment. The station controller can select any available GEM communication port in order to load a program and establish a virtual tester.

7.2.8 Multiple station controllers can address a single physical test system, with each station controller establishing its own virtual testers, until the physical limit for the test system is met.

7.3 *GEM Communication Synchronicity to Virtual and Physical Tester*

7.3.1 The GEM communication client interface must represent the actual real time state of the related virtual tester at all times.

7.3.2 Events from outside the GEM communication interface, such as local operator actions, can cause state transitions to occur.

7.3.3 Synchronization of the virtual tester to physical tester events (from any source) is required.

7.3.4 The methodologies and effectiveness of this virtual tester to physical tester synchronization is outside the scope of the TSEM and is the responsibility of the test vendor.

7.4 *GEM communication Sessions and Alarms*

7.4.1 Any alarm condition that impacts more than one virtual tester must be communicated to all impacted virtual testers.

7.5 *State Model Implications*

7.5.1 When the test system workstation is powered up, and the workstation operating system (O/S) successfully loads, (n) GEM communication port IDs will be established and made available for the station controller. Each GEM communication port represents a potential virtual tester. Each GEM communication port will have a fully independent process state model, which initializes into the GEM READY state.

7.5.2 The station controller can connect to and establish a GEM communication session with any of the available GEM communication ports. Once the station controller establishes a GEM communication session, the remote command START EXEC will load the tester executive and, if successful, cause the process state model for that GEM communication session or virtual tester to transition to the IDLE state. This transition is wholly independent of any other GEM communication sessions. The other GEM sessions remain in their original states.

Example: The host issues the first START EXEC to GEM communication port #1. The virtual tester transitions to INIT and starts the Tester Executive which in turn initializes all hardware. If successful, the virtual tester then transitions to IDLE. GEM communication ports 2 through (n) remain in a "logical" GEM READY state even though the physical test system has started the Tester Executive.

7.5.3 If a previous START EXEC remote command to a different GEM communication session on the same tester workstation has already started the tester executive, and the tester executive can support multiple virtual testers, then the new GEM communication session can transition to the IDLE state without starting a new tester executive.

7.5.4 The station controller can issue a STOP EXEC remote command to an active virtual tester. If another virtual tester is running in that test system, the virtual tester that received the remote command will transition to the GEM READY state, but leave the tester executive running. If no other virtual testers are currently running on that test system, the test system can shut down the tester executive**.**

7.6 *Physical Resources and Virtual Testers*

7.6.1 For the case of a multi-site virtual tester, each test-site should be uniquely identified without having to identify the test-head associated with that test-site.

7.6.2 Once a GEM communication session or virtual tester is in the IDLE state, the station controller can issue the PP-SELECT remote command. Until a PP-SELECT has been issued to a virtual tester, and the SETTING UP state is entered, the GEM communication session exists only as a communications path to a potential virtual tester.

7.6.3 In the SETTING UP state, the test system will determine whether the physical hardware resources required for that test program are available. If the resources are available, they are reserved for this virtual tester, the test program loads, SETTING UP process state is successful, and the virtual tester transitions to READY. If the physical hardware resources required for that test program are not available, the SETTING UP process state fails, the virtual tester has an ALARM, and it returns to the IDLE state. Before entering the IDLE state, all resources that have been reserved during SETTING UP must be freed.

7.6.4 Depending on the architecture of the test system, the hardware resources may or may not be tied to a specific virtual tester.

Example: A test system with three test-heads might require the use of virtual tester 1 to address test-head 1, virtual tester 2 to address test-head 2, and virtual tester 3 to address test-head 3. If a process-program requiring the resources of two test-heads is loaded on virtual tester 1, it might assign both test-head 1 and test-head 2 to that virtual tester. An attempt to load a program on virtual tester 2 would then fail, but a program might still be loaded to virtual tester 3.

7.6.5 If all test-sites on a test-head are disabled, the test-head itself may be considered to be disabled. If a test-head is disabled the test-head may be undocked from the unit handling equipment and the test-board may be removed from the test-head. Commands to enable test-sites on a test-head that is not connected to the test-board or is not docked should fail. Testing should be able to proceed on test-sites on other test-heads.

7.6.5.1 If a virtual tester includes test-sites on more than one test-head, disabling a test-head should not prevent testing from continuing on the test-sites on other test-heads.

8 Collection Event List

8.1 *Requirements*

8.1.1 All defined TSEM process state model state transitions are required collection events. In addition to defined process state model state transition collection events, the following collection events are also required.

Table 3 Additional Required Collection Events

Collection Event Name	*Collection Event Description*
TestBoardChange	A test-board is attached, removed, or changed.
DockStatusChange	The test-head is docked or undocked from the materials handler.
EnableSite	A previously disabled test-site is enabled.
DisableSite	A previously enabled test-site is disabled.

9 Variable Items

9.1 The purpose of this section is to define the list of variable items required by the TSEM. Values of these variables will be available to the host through collection event reports and host status queries.

9.2 *Requirements*

9.2.1 Variable items required by TSEM are categorized as follows:

- *Common Variables (CVs)* — Variables common to all testers.
- *Configuration-Specific Variables (CSVs)* — Variables associated with a specific configuration of the above equipment class.

9.2.2 Equipment constants have various uses in TSEM, including the following:

- Equipment offsets that match the performance of several pieces of equipment that would otherwise perform differently due to inherent manufacturing differences. Examples are home values and motion axis scaling factors.
- Setting the configuration of the equipment to allow for different material specifications, equipment options, material flows, frequency of automatic functions, etc. An example is yield check frequency.
- Managing optional machine features. Examples are constants that indicate whether optional features such as automated media stackers are present and control the configuration and function of these optional subsystems when they are present.
- Equipment Constants can be changed by the user but are not modified by the equipment. Its value can be queried by the host at any time. Equipment constants remain in effect until they are modified by the user.

9.2.3 Status Variables are valid at all times. A status variable may not be changed by the host but may be changed by the equipment or operator. The value of status variables may be queried by the host at anytime.

9.2.4 Data Variables are valid only upon the occurrence of specific collection events. Data variables may be queried by the host. An attempt to read a data variable item at the wrong time does not generate an error, but the data reported may not have relevant meaning.

9.2.5 *Data Item Requirements for Multi-Head, Multi-Site Equipment* — Some variables may change value depending on the processing state (whether or not resources are allocated to a virtual tester). See definition of Virtual Tester.

Example: If calibration is done separately on two test-heads, and a virtual tester has been allocated resources only from the more recently calibrated test-head, CalDate for that virtual tester may have a more recent date in the READY state than it did in the IDLE state.

9.3 *class, hard-bin, and soft-bin* — Equipment is to maintain variables that provide three levels of granularity for test results: class, hard-bin, and soft-bin. Classes, hard-bins, and soft-bins are defined within a process program, and their values are made available during events. When unit testing has completed, the process program is to determine the class, hard-bin, and soft-bin with which the unit is to be associated, based on the results of the testing.

9.4 *Variable Item Tables*

Table 4 Variable Item Table

Variable Name	*Category*	*Description*	*Format*	*Comments*
ActiveSites	CV	List of test-sites that are to be tested.	List of integer	
Address	CSV	Address of the reported vector.	Integer	Valid only in Datalog reports.
Channel	CV	Hardware channel or resource identifier.	Text	List by PinID.
ClassID	CV	Test result class number.	Integer	Valid in PROCESS states. List by test-site.
ClassName	CV	Test result class name.	Text	Valid at BIN-DATA-AVAILABLE state. List by ClassID.
ConfigInfo	CSV	Physical configuration information.	Format is determined by vendor.	Valid in all states.
Cycle	CV	Cycle count for this report.	Integer	Valid only in Datalog reports.
DatalogConfig	CV	Datalog Configuration.	Text	Valid in all states.
DatalogPlanName	CV	Datalog report plan name.	Text	Valid in all states.
DockingStatus	CSV	Information on handler/prober docking status.	Boolean	Valid in all states.
EnabledSites	CV	List of test-sites initially enabled at process program download.	List of integer	
EquipMake	CV	Tool Manufacturer.	Text	Valid in all states.
EquipSerialID	CV	Unique Equipment identifier.	Text	Valid in all states.
FunctionalResult	CSV	Vector of bits indicating pass or fail for each pin. Where 0 = Fail and 1 = Pass.	Binary	Valid only in Datalog reports.
HardBinID	CV	Test result hard-bin number.	Integer	Valid in PROCESS states. List by test-site.
HardBinName	CV	Test result hard-bin name.	Text	Valid at BIN-DATA-AVAILABLE Event. List by HardBbinID.
HighLimit	CSV	Higher limit for measurement. Multiple instances of this variable exist. See TEST-LIST report.	Float	List by TestID.
LowLimit	CSV	Lower limit for measurement. Multiple instances of this variable exist. See TEST-LIST report.	Float	List by TestID.

Variable Name	*Category*	*Description*	*Format*	*Comments*
NumberOfHeads	CV	The number of test-heads on the system.	Integer	
NumberOfPins	CV	The number of pins for each test-head.	Integer	
OperatorID	CV	Current Operator ID.	Text	Valid in all states.
Pass	CV	Where 0 = Fail and 1 = Pass.	Boolean	Valid only in Datalog reports.
PatternName	CSV	This must identify the location to which the vector address and cycle count are relative.	Text	Valid only in Datalog reports.
PinID	CV	Pin identifier used to identify the pin results in pin result reports.	Integer	List by PinID.
PPExecVersion	CV	Test program version.	Text.	
ProcessProgramID	CV	Process program identifier.	Text	Valid in the processing active states.
Range	CSV	Range used by the measurement unit.	Float	Valid only in Datalog reports.
RealResult	CSV	Measured value.	Float	Valid only in Datalog reports.
Signal	CV	Signal name.	Text	List by PinID.
SiteID	CV	Test-board test-site number for multi-site (parallel) testing.	Integer	
SoftBinID	CV	Test result soft-bin number.	Integer	Valid in PROCESS states. List by test-site.
SoftBinName	CV	Test result soft-bin name.	Text	Valid at BIN-DATA-AVAILABLE Event. List by SoftBinID.
SoftwareBuildID	CV	The build ID of this software list element. Multiple instances of this variable exist. See SOFTWARE-LIST report.	Text	
SoftwareName	CV	The name of this software list element (e.g. Solaris, IG9000, etc.) Multiple instances of this variable exist. See SOFTWARE-LIST report.	Text	
SoftwareType	CV	The type of this software list element (e.g. Operating System, Tool Control System, etc.) Multiple instances of this variable exist. See SOFTWARE-LIST report.	Text	
SoftwareVersion	CV	The version of this software list element. Multiple instances of this variable exist. See SOFTWARE-LIST report.	Text	
StartTestPortID	CSV	Start Test Source (i.e. hand, keyboard, host).	Text	Valid in all states.
TestBoardCalStatus	CV	Test-board calibration status. Where 0 = Fail and 1 = Pass.	Boolean	
TestBoardID	CV	ID of current test-board.	Text	Valid in all states.
TestBoardPosition	CV	A relative location of a test-site on a test-board.	Integer[2]	X and Y position. Site 1 is 0,0.
TestBoardSiteID	CV	A mapping of a test-site to a physical location on the tester.	List of SiteId TestBoardID TestBoardPosition	

Variable Name	*Category*	*Description*	*Format*	*Comments*
TestBoardStatus	CV	Test-board availability (1 = enabled, 0 = disabled).	Boolean	Valid in PROCESS states. List by test-site.
TestBoardType	CV	Type of fixture (test-board, probecard, cable, contactor, etc.).	Text	
TestHeadID	CV	The ID of the test-head.	Integer	Valid in READY state.
TestHeadStatus	CV	(2 = Not Available, 1 = enabled, 0 = disabled)	Enumerated	Valid in IDLE state.
TestID	CV	Test identifier used to identify test results. Multiple instances of this variable exist. See TEST-LIST report.	Integer	
TestInstance	CV	Test instance identifier. The TestID and TestInstance combined should be unique for one execution of the test program. They can be the same for all results from units tested in parallel by the same instance of a single test (possibly different for a serially applied test in a multi-site test program).	Integer	Valid only in Datalog reports.
TestName	CV	Test Name. Multiple instances of this variable exist. See TEST-LIST report.	Text	List by TestID.
TestSetup	CV	Setup conditions used for the test. Multiple instances of this variable exist. See TEST-LIST report.	Text	List by TestID.
TestStatus	CV	Test status of a specific test in the test program: 1 = Pass 2 = Fail 3 = Both	Enumerated	Valid only in Datalog reports. When defining datalog reports, "both" means that the same report will be generated for either pass or fail.
TestTime	CV	Test execution time in seconds.	Float	Valid only in Datalog reports.
TestType	CV	1 = Parametric Test (real value result) 2 = Functional test (vector of pass/fail results) 3 = Text test (arbitrary text string as result) Multiple instances of this variable exist. See TEST-LIST report.	Enumerated	List by TestID.
TextResult	CSV	Arbitrary string of text.	Text	Valid only in Datalog reports.
UnitID	CV	Unit Serial Number.	Text	Valid in all states. List by test-site.
Units	CV	Electrical units to be measured. Multiple instances of this variable exist. See TEST-LIST report.	Text	List by TestID.
VirtualConfig	CSV	Current Virtual Configuration listing all test-sites used.	List of TestBoardSiteID TestHeadId	Valid in PROCESS states. List all possible test-sites for current virtual tester.

Table 5 Variable Items Dependent on State

The values of these variables may change based on processing state (See Section 9.2.5).

Variable Name	*Category*	*Description*	*Format*	*Comments*
AvailableResources	CSV	Returns all available resources (resources not currently assigned to any virtual tester).	Format is determined by vendor.	Valid in all states. Most likely a subset of ConfigInfo.
CalDate	CV	Date and time of last successful calibration.	YYYYMMDDhhmmsscc	Valid in all states.
CalFixtureID	CV	ID of calibration fixture in current configuration.	Text	Valid in all states.
CalInterval	CV	Time limit between calibrations.	YYYYMMDDhhmmsscc	Valid in all states.
CalStatus	CV	Status of last calibration.	Boolean	Valid in all states.
DiagDate	CV	Date of last diagnostic execution.	YYYYMMDDhhmmsscc	Valid in all states.
DiagFixtureID	CV	ID of diagnostic fixture in current configuration.	Text	Valid in all states.
DiagInterval	CV	Time limit between diagnostics.	YYYYMMDDhhmmsscc	Valid in all states.
DiagStatus	CV	Status of last diagnostic.	Boolean	Valid in all states.

10 TSEM-Required Reports

10.1 TSEM requires two types of reports. The first type is the collection event report. Collection event reports are associated with process state model transition events or other non-transition collection events as described in Section 8.1.1 . The second type is the datalog report. Datalog reports are not specifically associated with a collection event and are intended to provide the host with test measurement result data.

10.2 *User-configured Reports* — The host shall be provided a mechanism to configure additional reports of either type. The method for defining either type of report and associating them with either collection events or datalog requirements shall be the same.

10.3 *Report Messages* — Collection event reports and datalog reports shall be provided to the host via two separate message services. This requirement is designed to prevent the larger datalog reports from impacting the execution speed of the process state model.

10.4 *Collection Event Reports* — The following reports are required as pre-configured collection event reports. The specific format of the report should follow the format provided by the collection event message definition.

Table 6 Collection Event Reports

Report Label	*Description*	
SETUP-REPORT	Must be valid	On process state transition 3. In READY, EXECUTING and PAUSE states.
	Description	Provides information on the general process program and test hardware setup.
	Parameters	TestHeadID
		DockingStatus
		TestBoardID
		PPExecName
		PPExecVersion

Report Label	*Description*	
BIN-DATA-REPORT	Must be valid	On process state transition 5. In READY and PAUSE States.
	Description	Provides test result information.
	Parameters	SiteID
		UnitID
		HardBinID
		SoftBinID
		ClassID
SITE-STATUS-REPORT	Must be valid	With the EnableSite or DisableSite collection event. In READY, EXECUTING and PAUSE states.
	Description	Used to notify the host that a test-site has been enabled or disabled.
	Parameters	EnabledSites
TEST-BOARD-REPORT	Must be valid	With the TestBoardChange collection event. In PROCESSING-ACTIVE, IDLE, and IDLE WITH ALARMS states.
	Description	Notifies host that the test-board has been attached, removed or changed.
	Parameters	TestHeadID
		DockingStatus
		TestBoardID
DOCK-STATUS-REPORT	Must be valid	With the DockStatusChange collection event. In PROCESSING ACTIVE, IDLE, and IDLE WITH ALARMS states.
	Description	Notifies host that the test-head has been docked or undocked from the materials handler.
	Parameters	TestHeadID
		DockingStatus
DATA-VARIABLE-IDS	Must be valid	On process state transition 3. In READY, EXECUTING and PAUSE states.
	Description	List of data variable item (DV) names and IDs.
	Parameters	Data Variable Name
		Data Variable ID
TEST-LIST	Must be valid	On process state transition 3. In READY, EXECUTING and PAUSE states.
	Description	This report relates the TestId variable to the detailed description of the test. This allows other tests to communicate the TestId variable without the need to communicate the detailed test information.
	Parameters	TestID
		TestName
		TestType
		LowLimit
		HighLimit
		Units
		TestSetup
PIN-LIST	Must be valid	On process state transition 3. In READY, EXECUTING and PAUSE states.
	Description	This report associates each PinID with the corresponding Signal name and tester hardware Channel ID. For multi-site configurations, the PinID and Signal parameters will be the same for multiple Channel parameters.
	Parameters	PinID
		Channel
		Signal

Report Label	*Description*	
CLASS-LIST	Must be valid	On process state transition 3. In READY, EXECUTING and PAUSE states.
	Description	The ClassID is a numeric value which identifies a group of hard-bins.
	Parameters	ClassID
		ClassName
HARDBIN-LIST	Must be valid	On process state transition 3. In READY, EXECUTING and PAUSE states.
	Description	A report of hard-bins defined and used by the process program.
	Parameters	HardBinID
		HardBinName
		ClassID
SOFTBIN-LIST	Must be valid	On process state transition 3. In READY, EXECUTING and PAUSE states.
	Description	A report of soft-bins defined and used by the process program.
	Parameters	SoftBinID
		SoftBinName
		HardBinID
		ClassID

10.5 *Datalog Reports* — The following reports are required as preconfigured datalog reports. The format for the report shall be as defined in this table.

Table 7 Datalog Reports

Report Label		*Description*
FUNCTIONAL-TEST-RESULT	Availability	Upon completion of a Functional Test.
	Description	Results for a Functional test.
	Format	List, 6 1. SiteID 2. TestID 3. TestInstance 4. Pass 5. TestTime 6. List, n (n = # of results reported) 1. List, 4 1. PatternName_1 2. Address_1 3. Cycle_1 4. $\text{FunctionalResult}_1$ … n. List, 4 1. PatternName_n 2. Address_n 3. Cycle_n 4. $\text{FunctionalResult}_n$

Report Label		*Description*
PARAMETRIC-TEST-RESULT	Availability	Upon completion of a Parametric Test.
	Description	Result of a Parametric test.
	Format	List, 6 1. SiteID 2. TestID 3. TestInstance 4. Pass 5. TestTime 6. List, n (n = # of results reported) 1. List, 3 1. $PinId_1$ 2. $RealResult_1$ 3. $Range_1$ ... n. List, 3 1. $PinId_n$ 2. $RealResult_n$ 3. $Range_n$
TEXT-TEST-RESULT	Availability	Upon completion of a Text Test.
	Description	Result of a Text test.
	Format	List, 6 1. SiteID 2. TestID 3. TestInstance 4. Pass 5. TestTime 6. List, n (n = # of results reported) 1. $TextResult_1$ 2. $TextResult_2$ … n. $TextResult_n$
BRIEF-RESULT	Availability	Upon completion of any kind of Test.
	Description	Generalized test result.
	Format	List, 5 1. SiteID 2. TestID 3. TestInstance 4. Pass 5. TestTime

Report Label		*Description*
DEVICE-RESULT	Availability	End of Test
	Description	Unit level result. One report is linked to the End of Test event. This one report contains information for all test-sites tested.
	Format	List, c (c = # of SiteIDs) 1. List, 4 1. $SiteID_1$ 2. $Pass_1$ 3. $SoftBinID_1$ … c. List, 4 1. $SiteID_c$ 2. $Pass_c$ 3. $SoftBinID_c$

11 Process Program Management

11.1 *Process Program Requirements*

11.1.1 The GEM requirements for process program management must be fully supported for this class of equipment. The following is also required. The process program must have a structure that enables the user to build process programs with default conditions that can be overridden at run time via parameters specified in the PP-SELECT remote command.

- When a process program is downloaded to the equipment by the host, the process program syntax shall be verified by the equipment.
- The downloaded process parameters shall be checked for type and range before execution. An error message must be generated from the tester if the process program parameters are outside the range of the machine capability.
- The contents of the downloaded process program body (PPBODY) may contain the explicit parameters and data necessary for the runtime process program, or the body may contain reference information (i.e., PATH location) on where the explicit data is stored. In the latter case, it is required that the equipment combines or compiles the reference data prior to the verification step.

11.2 Diagnostic and calibration routines are considered process programs and must be verified and validated the same as a typical process program.

12 Alarm Management

12.1 *Alarm Definition*

12.1.1 The TSEM requires a broader definition of alarms than is provided by SEMI E30. The alarm definitions provided by SEMI E5 provide this broader definition. SEMI E5 provides for eight (8) categories of alarms:

1. *personal safety* — Condition may be dangerous to people.
2. *equipment safety* — Condition may harm equipment.
3. *parameter control warning* — Parameter variation outside of preset limits — may harm product.
4. *parameter control error* — Parameter variation outside of reasonable control limits — may indicate an equipment malfunction.
5. *irrecoverable error* — Intervention required before normal use of equipment can resume.
6. *equipment status warning* — An unexpected condition has occurred, but operation can continue.
7. *attention flags* — A signal from a process program indicating that a particular step has been reached.
8. *data integrity* — A condition that may cause loss of data.

12.1.2 It will be the equipment's responsibility to categorize the alarm. Some alarm conditions may cause more than one type of alarm to be issued. For example, a parameter control error for over voltage may also trip a protective device that makes the alarm irrecoverable without some intervention.

12.2 *Relationship to ALARM PAUSED Process State*

12.2.1 The relationship between an alarm condition and the ALARM PAUSED process state should be defined by the equipment manufacturer on a case-by-case basis along these guidelines.

1. An alarm does not necessarily require a process state change to ALARM PAUSED even though it may ultimately prove to be the root cause of that transition at a later time.
2. Alarms that force immediate processing stoppage for the primary function of the tool should trigger a transition to ALARM PAUSED.
3. Most alarms in categories 1–3 will cause a transition to ALARM PAUSED.

12.2.2 A machine with currently set alarms may PAUSE instead of ALARM PAUSED if none of the currently set alarms were the immediate trigger for the transition.

13 Remote Commands

13.1 The purpose of this section is to identify remote commands, command parameters, and valid commands versus states in the processing state models.

13.2 *Requirements*

- The equipment must support the GEM-required remote commands. (Some of the SEMI E30-required remote commands are restated here to define TSEM-specific requirements.)
- All the remote commands defined by TSEM are required.
- The alphanumeric strings defined by TSEM for remote commands and command parameters are required.
- If additional remote commands are supported, then the “Remote Command versus Valid States” matrix must be generated for these additional commands. Place an “X” in the table for each state in which a given command is valid.

13.3 *Remote Commands and TSEM Process Model Mapping*

13.3.1 Table 9 illustrates the relationship between remote commands and states of the TSEM processing state model. An “X” indicates that a command is valid for use in this state. If a remote command is attempted during a non-valid state, the equipment would reject the remote command.

13.4 *Remote Command Descriptions*

13.4.1 *ABORT* — This command terminates the current processing. ABORT makes no guarantee about completion of the current test(s).

13.4.2 *PAUSE* — This command transitions the tester to the PAUSING process state when the current test(s) completes processing.

13.4.3 *RESUME* — This command resumes processing from the point where the process was PAUSED. Process program variable parameters can be specified in this command that modify the default values for these variable parameters in the process program. Process program verification (CHECKING state) must occur when variable parameters accompany this command.

13.4.4 *PP-SELECT* — This command instructs the tester to copy the indicated Process Program from non-volatile storage to the tester’s process program execution area. Process program variable parameters can be specified in this command that modify the default values for these variable parameters in the process program. Process program verification must occur when variable parameters accompany this command.

13.4.5 *DEFINE-DATALOG-PLAN* — This command defines and names a datalog plan. The START command may subsequently activate the datalog plan by name.

13.4.6 *START* — This command is only available to the host or operator when a process program has been selected and the tester is in the READY processing state. The START command instructs the tester to initiate processing. Parameters can be specified in this command.

13.4.7 *STOP* — This command completes the current test(s), stops in a safe condition, and returns to the IDLE processing state. STOP has the intent of bringing about a normal termination after completion of the current test(s).

13.4.8 *START EXEC* — This command instructs the tester to start the tester executive.

13.4.9 *STOP EXEC* — This command instructs the tester to stop the tester executive.

13.4.10 *ENABLE-SITE* — This command instructs the tester to persistently re-enable a test-site previously disabled via the DISABLE-SITE remote command. Parameters can be specified in this command.

13.4.11 *DISABLE-SITE* — This command instructs the tester to persistently disable a test-site originally enabled via the PP-SELECT remote command. Parameters can be specified in this command.

13.5 *Associated Remote Command Parameters*

Table 8 Remote Commands

Command Name	*Parameter Name*	*Opt./Req.*	*Parameter Format*	*Description*
ABORT	None			
DISABLE-SITE	DISABLESITELIST	REQ	List, s (s = # of test-sites) 1. (test-site 1) 2. (test-site 2) s. (test-site s)	ID of test-site to be disabled.
ENABLE-SITE	ENABLESITELIST	REQ	List, s (s = # of test-sites) 1. (test-site 1) 2. (test-site 2) s. (test-site s)	ID of test-site to be enabled.
PAUSE	None			
PP-SELECT	DEFINEDSITELIST	OPT	List, s (s = # of test-sites) 1. (test-site 1) 2. (test-site 2) s. (test-site s)	Establish the initial default set of test-sites.
	PROCESSPARAMETER	OPT	User specific	
	PROCESSPROGRAMID	REQ	Text	Identifier of the process program to be executed.
RESUME	PROCESSPARAMETER	OPT	User specific	

Command Name	*Parameter Name*	*Opt./Req.*	*Parameter Format*	*Description*
START	SELECTEDSITELIST	OPT	List, s (s = # of test-sites) 1. (test-site 1) 2. (test-site 2) s. (test-site s)	Provides single execution override of test-sites enabled/disabled by remote commands ENABLE-SITE and DISABLE-SITE.
	DATALOGDATASETS	OPT	List, n (n - # of datasets) 1. List, 2 1. Dataset Name$_1$ 2. List, p (Zero-length list = all test-sites) 1. SiteID$_{11}$ … p. SiteID$_{1p}$ n. List, 2 1. Dataset Name$_n$ 2. List, q (Zero-length list = all test-sites) 1. SiteID$_{n1}$ … q. SiteID$_{nq}$	Datalog output can be directed to separate datasets on a per-test-site basis.
	DATALOGBYTESTTYPE	OPT	List, b (b = # of (TestType, TestStatus) pairs being assigned reports.) 1. List, 3 1. TestType$_1$ 2. TestStatus$_1$ 3. ReportID$_1$ (If omitted, the default report is assigned.) … b. List, 3 1. TestType$_b$ 2. TestStatus$_b$ 3. ReportID$_b$	Specifies datalog options by general TestType.
	DATALOGBYTESTID	OPT	List, c (c = # of (TestId, TestStatus) pairs being assigned reports.) 1. List, 3 1. TestID$_1$ 2. TestStatus$_1$ 3. ReportID$_1$ … c. List, 3 1. TestID$_c$ 2. TestStatus$_c$ 3. ReportID$_c$	Specifies datalog options for specific test ids.
	DATALOGOPTION	OPT	Enumerated (0:No Log, 1:Log Pass, 2:Log Failures, 3: Log All)	Selects the condition when datalog is sent.
	DATALOGPLANNAME	OPT	Text	Turns on datalogging according to a predefined plan. This parameter overrides all other individual datalog setting options in a START command.

Command Name	*Parameter Name*	*Opt./Req.*	*Parameter Format*	*Description*
DEFINE-DATALOG-PLAN	DATALOGPLANNAME	REQ	Text	Name of datalog plan.
	DATALOGDATASETS	OPT	List, n (n - # of datasets) 1. List, 2 1. Dataset Name$_1$ 2. List, p (Zero-length list = all test-sites) 1. SiteID$_{11}$ … p. SiteID$_{1p}$ n. List, 2 1. Dataset Name$_n$ 2. List, q (Zero-length list = all test-sites) 1. SiteID$_{n1}$ … q. SiteID$_{nq}$	Datalog output can be directed to separate datasets on a per-test-site basis.
	DATALOGBYTESTTYPE	OPT	List, b (b = # of (TestType, TestStatus) pairs being assigned reports.) 1. List, 3 1. TestType$_1$ 2. TestStatus$_1$ 3. ReportID$_1$ (If omitted, the default report is assigned.) … b. List, 3 1. TestType$_b$ 2. TestStatus$_b$ 3. ReportID$_b$	Specifies datalog options by general TestType.
	DATALOGBYTESTID	OPT	List, c (c = # of (TestId, TestStatus) pairs being assigned reports.) 1. List, 3 1. TestID$_1$ 2. TestStatus$_1$ 3. ReportID$_1$ … c. List, 3 1. TestID$_c$ 2. TestStatus$_c$ 3. ReportID$_c$	Specifies datalog options for specific test ids.
	DATALOGOPTION	OPT	Enumerated (0:No Log, 1:Log Pass, 2:Log Failures, 3: Log All)	Selects the condition when datalog is sent.
START EXEC	None			
STOP	None			
STOP EXEC	None			

13.6 *Enable/Disable Site Hierarchy*

13.6.1 Test-sites can be enabled and disabled either persistently or for a single execution of the process program through the use of the PP-SELECT, ENABLE-SITE, DISABLE-SITE, and START remote commands. The relationship hierarchy for these commands is as follows:

1. The ENABLE-SITE parameter on the PP-SELECT remote command establishes the initial default set of test-sites.

2. The SELECT-TEST-SITE parameter on the START remote command establishes a single execution subset of the initial default set of test-sites established with the PP-SELECT remote command.
3. The ENABLE-SITE and DISABLE-SITE remote commands provide for a persistent override of the initial default set of test-sites to be established with the PP-SELECT remote command.
   - The DISABLE-SITE remote command will persistently disable a test-site originally enabled via the ENABLE-SITE parameter of the PP-SELECT remote command.
   - The ENABLE-SITE remote command will persistently re-enable a test-site that was disabled via the DISABLE-SITE remote command.
   - The ENABLE-SITE remote command may not enable a test-site that was not initially enabled via the ENABLE-SITE parameter on the PP-SELECT remote command.

13.7 *Remote Commands and TSEM Process Model Mapping*

13.7.1 Table 9 illustrates the relationship between remote commands and states of the TSEM processing state model. An "X" indicates that a command is valid for use in this state. If a remote command is attempted during a non-valid state, the equipment would reject the remote command.

Table 9 Remote Commands vs. Process States

COMMAND / *PROCESSING STATE*	STOP	START	RESUME	DEFINE-DATALOG-PLAN	PP-SELECT	DISABLE-SITE	ENABLE-SITE	PAUSE	STOP EXEC	START EXEC	ABORT
GEM READY										X	
INIT											
INIT WITH ALARMS											
IDLE					X				X		
IDLE WITH ALARMS									X		
PROCESSING ACTIVE											
PROCESS	X							X			X
SETTING UP	X							X			X
READY	X	X		X		X	X	X			X
EXECUTING	X							X			X
PAUSE	X										X
PROCESS PAUSE	X										X
PAUSING	X		X								X
PAUSED	X		X	X		X	X				X
CHECKING	X										X
ALARM PAUSED	X										X
ABORTING											
STOPPING											X

14 Additional SEMI E30 Requirements

The purpose of this section is to specify any GEM additional capabilities that are required to be supported by this class of equipment.

14.1 *Requirements* — The following GEM additional capabilities required by TSEM are:

- Establish Communications,
- Dynamic Event Report Configuration,
- Variable Data Collection,
- Status Data Collection,
- Alarm Management,
- Remote Control,
- Equipment Constants,
- Process Program Management,
- Equipment Terminal Services,
- Clock,
- Spooling, and
- Control (Host-Initiated).

15 TSEM Unique Capabilities

15.1 The purpose of this section is to specify additional capabilities required for the TSEM that are unique to this class of equipment.

15.2 *Host Access to Test Results Information* — TSEM requires the equipment manufacturer to make test datalog information available to the host. Because equipment configurations vary for datalog content and format, the equipment manufacturer must document the format and content of datalog information used by the equipment and make that information available across the communications interface. A test data acquisition interface and mechanism for management of test data collection is described below.

15.3 *Datalog Plan* — The datalog plan describes a method by which the host can request test result data to be communicated in an automated fashion by the equipment. It specifies the set of data in which Host is interested in and that the equipment should send off-tool. It includes a mechanism for organizing related data into groups or categories to make it more straightforward for Host to enable or disable a large number of data sources, and to allow Host to organize related data into groups according to their purpose.

15.4 *Defining a Datalog Plan* — The remote command DEFINE-DATALOG-PLAN will assign a name to a set of reports and datasets. Named datalog plans defined in this manner are undefined when the tester leaves the PROCESSING ACTIVE state. There must be one predefined datalog plan called "NONE" that will disable all datalog reports.

15.5 *Test Data Types* — The types of test data that are commonly collected are described in the following sections, grouped by the type of test that generated the data. This list, which may not be exhaustive, represents the minimum required data to be supported by the test equipment. In datalog plans, the TestType can be used to enable or disable an entire group of data to be communicated to the host; see *TestType* data variable for enumerated values of TestTypes. Upon loading of the test recipe or process program, the test equipment should have knowledge of all test steps as well as their respective TestTypes.

15.5.1 *Functional Test Data* — Test data associated with a functional test, such as pattern name and the functional test result. Functional test results are normally a vector of pass/fail values.

15.5.2 *Parametric Test Data* — Test data associated with a parametric test. Parametric test results are normally a measured value from within a range.

15.5.3 *Text Test Data* — Test data associated with a text test. Text test results contain an arbitrary string of text.

15.6 *Test Steps* — Upon successful completion of process program load, the tester should have full knowledge of all the functional, parametric, etc. test steps defined by the process program. Each test step is uniquely identified by a test step ID and associated with a TestType. Information on all test steps defined must be available to Host via defined reports and can be attached to program load complete collection event, see TEST-LIST report.

15.7 *Collection Plan Definition* — Definition of a datalog plan to the equipment specifies the exact test data to be collected and/or communicated to the host and specifies a way to group data. This plan will remain effective until changed by host via another plan. Below outlines the general structure of the collection plan.

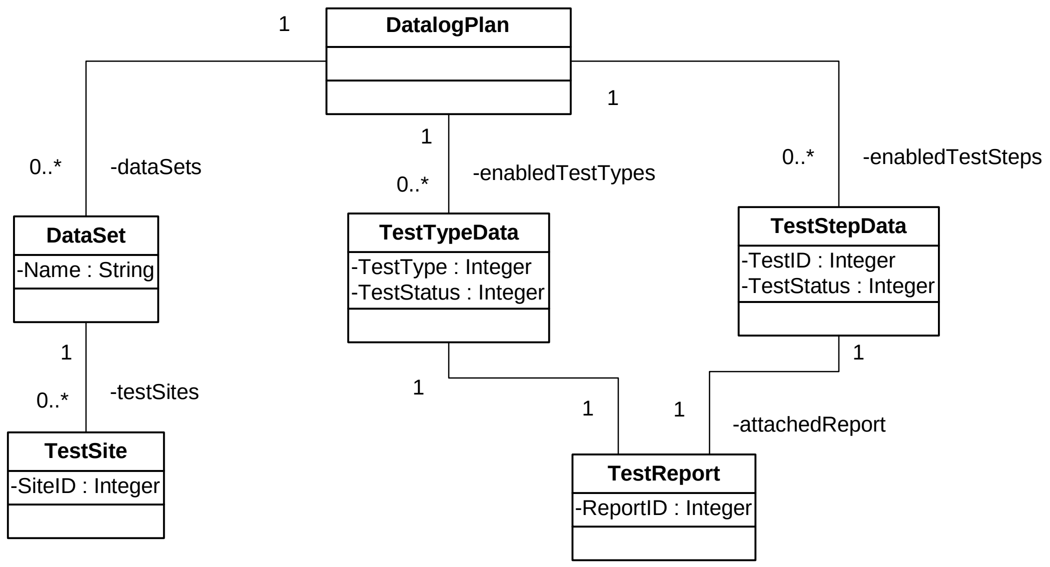

Figure 2
Datalog Plan

15.7.1 *Datalog Plan* — The plan specifies what TestTypes, test steps are enabled for data collection and describes the data to be collected. It also specifies a way to group the data into data sets based on test-site.

15.7.2 *Dataset* — The Dataset defines a grouping of data based on test-site of unit data. One or more test-sites may be specified for each dataset. If no test-site is specified, all test-sites is assumed. Each Dataset is associated with a name.

15.7.3 *TestSite* — TestSite specifies the processing test-site of the unit where test data was produced.

15.7.4 *TestTypeData* — TestTypeData defines the data to be collected for a particular TestType (functional, parametric etc) via an attached data report definition.

15.7.4.1 *TestType* — Specifies the type of test for which to collect data on via the attached report definition.

15.7.4.2 *TestStatus* — Specifies the pass/fail status of test for which data will be communicated to host.

15.7.5 *TestStepData* — TestStepData defines the data to be collected for a particular test via an attached data report definition. TestID specifies the ID of the test as provided by the TEST-LIST report, and TestStatus specifies pass/fail condition for which data will be communicated to host.

15.7.6 *TestReport* — TestReport defines the structured format and the data variables to communicate to host. The ReportID uniquely identifies the report defined in the equipment.

15.8 *Test Data Reporting* — Test data reporting is enabled by specifying a DATALOG-PLAN parameter in the START command or by defining an unnamed datalog plan in the STARTcommand using the DATALOG-DATASETS, DATALOG-BY-TESTTYPE, DATALOG-BY-TESTID, and DATALOG-OPTIONS parameters. If no datalog plan is specified in the START command, the last supplied datalog plan is used. Initially (after a PP-SELECT), all test data reporting will be disabled (using datalog plan “NONE”). The unnamed plan defined in the START command will be undefined as soon as another datalog plan is activated.

15.8.1 Sample test data reporting can be done by setting the DATALOG-PLAN to “NONE” and then, every once in a while, setting it to another datalog plan that generates the desired reports.

15.8.2 If limited test data reporting is desired at all times, but full test data reporting is required on a sample basis, two datalog plans can be defined and each START command can indicate the appropriate datalog plan.

15.9 *Collection Plan Activation* — The collection plan is communicated to equipment as parameters to the START remote command, with one parameter specifying the Dataset, TestTypeData and TestStepData definitions respectively. An additional parameter is also included to globally enable or disable all test data collection for process program. The parameters are DATALOG-DATASETS, DATALOG-BY-TESTTYPE, DATALOG-BY-TESTID, and DATALOG-OPTIONS. Please see START remote command parameters in Table 8 for format details.

15.10 *Test Data Acquisition* — This section specifies an interface to allow transfer of data sets from equipment to host. It defines at a conceptual level the messaging services, and does not define the actual messaging protocol. Information on structure of the message, how data is represented, or how the message is exchanged will be defined in a separate document, which maps the services to a specific protocol (such as SECS-II).

15.10.1 *Test Data Transfer Communications* — The required messages and the exchange sequence for a normal transfer of data sets between two systems are presented below in Figure 3.

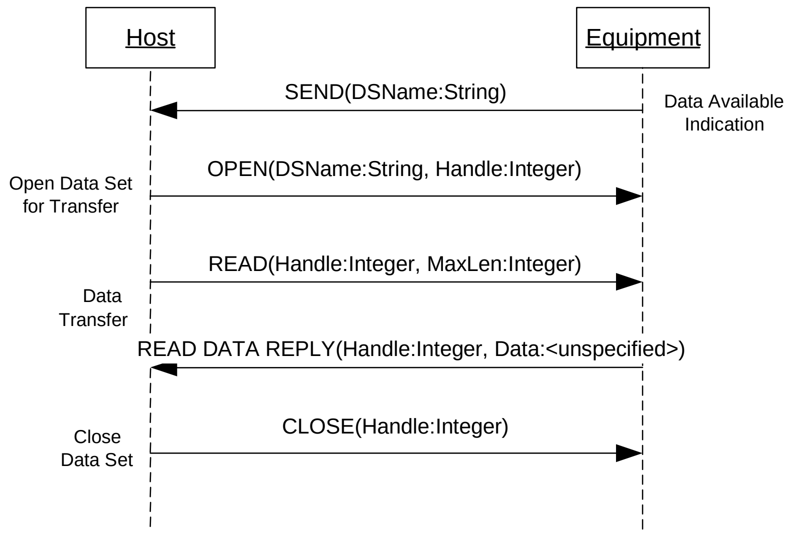
Host
Equipment
SEND(DSName:String)
Data Available Indication
OPEN(DSName:String, Handle:Integer)
Open Data Set for Transfer
READ(Handle:Integer, MaxLen:Integer)
Data Transfer
READ DATA REPLY(Handle:Integer, Data:<unspecified>)
CLOSE(Handle:Integer)
Close Data Set

Figure 3
Test Data Transfer Message Flow

15.10.1.1 *SEND* — Used by equipment to indicate to host that data is available for a data set and request host system to read it. The data set is identified by its name in the argument.

15.10.1.2 *OPEN* — Host requests that a certain data set be opened for transfer. A handle is provided unique to this session to be used in subsequent transfer requests.

15.10.1.3 *READ* — Host requests data be transferred for an open data set, identified by the handle. A max length is provided to indicate to equipment the maximum byte length of the data to be sent.

15.10.1.4 *READ DATA REPLY* — Equipment reply to a READ request. A block of data no bigger than the max length specified in the READ request is attached to the reply message. Handle identifies the open data set of the sent data.

15.10.1.5 *CLOSE* — Closes a data transfer session and frees the handle.

15.10.1.6 *RESET* — Closes all open data sets.

15.10.2 *Synchronization* — In order to synchronize the two systems during recovery situations, a RESET message is defined that can be used by either Host or Equipment to unconditionally close all open data sets.

15.10.3 This specification must be used in conjunction with a protocol mapping specification in order to get a full definition of the test data acquisition interface.

SEMI E122.1-0703
SPECIFICATION FOR SECS-II PROTOCOL FOR TESTER SPECIFIC EQUIPMENT MODEL (TSEM)

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 April 11, 2003. Initially available at www.semi.org May 2003; to be published July 2003.

1 Purpose

1.1 This document maps the services and data of SEMI E122 to SECS-II streams and functions, and data definitions.

2 Scope

2.1 This is a specification covering equipment supporting automated control job management.

2.2 This document applies to all implementations of SEMI E122 that use the SECS-II message protocol (SEMI E5). Compliance to this standard requires compliance to both SEMI E122 and SEMI E5.

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

3 Limitations

3.1 This specification applies to semiconductor equipment that also use SEMI E30 GEM standard.

4 Referenced Standards

4.1 *SEMI Standards*

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

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

SEMI E122 — Standard for Tester Equipment Specific Equipment Model (TSEM)

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

5 Services Message Mapping

5.1 This section shows the specific SECS-II streams and functions that shall be used for SECS-II implementation of the services defined in SEMI E122, as well as the parameter mapping for data attached to services.

5.2 *Services Message Mapping*

5.2.1 Table 1 defines the relationships between SEMI E122 services and SECS-II messages. Mapping of service parameters to the SECS-II data items is provided in a separate table. Conventions and definitions of table fields are as described below.

5.2.2 *Service Name* — Name of the service or remote command defined in SEMI E122.

5.2.3 *Stream, Function* — Specifies the SECS-II stream and function (SxFx) mapped to the service messages. Following convention of SECS-II, request and notification messages are mapped to the odd-numbered functions and response or acknowledgement messages are mapped to the corresponding even-numbered functions.

5.2.4 *SECS-II Message Name* — Name of the SECS-II message.

Table 1 Services Message Mapping Table

Service Name	*Stream, Function*	*SECS-II Message Name*
ABORT	S2,F41/F42	Host Command Send/Acknowledge
DEFINE-DATALOG-PLAN	S2,F49/F50	Enhanced Remote Command Send/Acknowledge
DISABLE-SITE	S2,F49/F50	Enhanced Remote Command Send/Acknowledge
ENABLE-SITE	S2,F49/F50	Enhanced Remote Command Send/Acknowledge
PAUSE	S2,F41/F42	Host Command Send/Acknowledge
PP-SELECT	S2,F49/F50	Enhanced Remote Command Send/Acknowledge
RESET-SITE-CNT	S2,F41/F42	Host Command Send/Acknowledge
RESUME	S2,F41/F42	Host Command Send/Acknowledge

Service Name	*Stream, Function*	*SECS-II Message Name*
START	S2,F49/F50	Enhanced Remote Command Send/Acknowledge
START-EXEC	S2,F41/F42	Host Command Send/Acknowledge
STOP	S2,F41/F42	Host Command Send/Acknowledge
STOP-EXEC	S2,F41/F42	Host Command Send/Acknowledge

5.3 *Data Collection Service Mapping*

5.3.1 Table 2 defines the relationships between SEMI E122 online test data collection services and SECS-II messages. These services maps to a subset of the SECS-II stream 13 functions, further usage and formatting details are as defined in stream 13 section of SEMI E5.

Table 2 Data Collection Service Message Mapping Table

Service Name	*Stream, Function*	*SECS-II Message Name*
SEND	S13,F1/F2	Send Data Set Send/Acknowledge
OPEN	S13,F3/F4	Open Data Set Request/Acknowledge
READ	S13,F5/F6	Read Data Set Request/Acknowledge
CLOSE	S13,F7/F8	Close Data Set Send/Acknowledge
RESET	S13,F9/F10	Reset Data Set Send/Acknowledge

5.4 *Event Message Mapping*

5.4.1 Table 3 defines the relationships between SEMI E122 collection events and SECS-II messages. Conventions and table field definitions similar to mapping table for the services, here Event Name specifies the event defined in SEMI E122.

5.4.2 Format and structure are as defined in SEMI E30 (GEM), including attached reports.

Table 3 Event Message Mapping Table

Event	*Stream, Function*	*SECS-II Message Name*
All state model transitions and all events defined in SEMI E122.	S6F11/12	Event Report Send/Acknowledge

5.5 *Service Parameter Mapping*

5.5.1 Table 4 defines the relationships between SEMI E122 service parameters and SECS-II data definitions or parameter fields in the mapped streams and functions. Parameters for acknowledgements or responses will follow SECS-II specification for that stream and function, no service specific error codes are defined in this specification. Descriptions of each table column are described below.

5.5.2 *Service* — Specifies the service whose parameters are described.

5.5.3 *SECS-II Field* — Specifies the SECS-II data item or message parameter used by the service. In this specification, the DATAID and OBJSPEC fields are unspecified for services using the S2F49 SECS-II message. Their values are to be ignored or used for implementation specific purposes.

5.5.4 *Values* — Value or SECS II format for the specified field. Formats are specified using the SML notation as defined in appendices of SEMI E30 (GEM).

5.5.5 *Req.* — Indicates whether the specified field or parameter is required or not. If an optional CPNAME parameter is used, the corresponding CPVAL must also be given.

Table 4 Service Parameter to SECS-II Data Items Mapping

Service	*SECS-II Field*	*Values*	*Req.*	*Description*
ABORT	RCMD	"ABORT"	Y	
DEFINE-DATALOG-PLAN	RCMD	"DEFINE-DATALOG-PLAN"	Y	
	CPNAME$_1$	"DATALOGPLANNAME"	Y	DATALOGPLANNAME parameter.
	CPVAL$_1$	A[80]	Y	Name of datalog plan.
	CPNAME$_2$	"DATALOGDATASETS"	N	DATALOGDATASETS parameter.
	CPVAL$_2$	L, n (n - # of datasets) 1. L, 2 1. A[80] 2. L, p 1. U4 … p. U4 n. L, 2 1. A[80] 2. L, q 1. U4 … q. U4	N	Datalog output can be directed to separate datasets on a per-test-site basis. L, n (n - # of datasets) 1. L, 2 1. Dataset Name$_1$ 2. L, p (Zero-length list = all test-sites) 1. SiteID$_{11}$ … p. SiteID$_{1p}$ n. L, 2 1. Dataset Name$_n$ 2. L, q (Zero-length list = all test-sites) 1. SiteID$_{n1}$ … q. SiteID$_{nq}$
	CPNAME$_3$	"DATALOGBYTESTTYPE"	N	DATALOGBYTESTTYPE parameter
	CPVAL$_3$	L, b 1. L, 3 1. U4 2. U4 3. U4 … b. L, 3 1. U4 2. U4 3. U4	N	Specifies datalog options by general TestType. L, b (b = # of (TestType, TestStatus) pairs being assigned reports.) 1. L, 3 1. TestType$_1$ 2. TestStatus$_1$ 3. ReportID$_1$ (If omitted, the default report is assigned.) … b. L, 3 1. TestType$_b$ 2. TestStatus$_b$ 3. ReportID$_b$
	CPNAME$_4$	"DATALOGBYTESTID"	N	

Service	*SECS-II Field*	*Values*	*Req.*	*Description*
	CPVAL$_4$	L, c 1. L, 3 1. U4 2. U4 3. U4 … c. L, 3 1. U4 2. U4 3. U4	N	Specifies datalog options for specific test ids. L, c (c = # of (TestId, TestStatus) pairs being assigned reports.) 1. L, 3 1. TestID$_1$ 2. TestStatus$_1$ 3. ReportID$_1$ … c. L, 3 1. TestID$_c$ 2. TestStatus$_c$ 3. ReportID$_c$
	CPNAME$_5$	"DATALOGOPTION"	N	DATALOGOPTION parameter.
	CPVAL$_5$	U4	N	Turns datalog on/off. (0:No Log, 1:Log Pass, 2:Log Failures, 3: Log All)
DISABLE-SITE	RCMD	"DISABLE-SITE"	Y	
	CPNAME$_1$	"DISABLESITELIST"	Y	DISABLESITELIST parameter.
	CPVAL$_1$	L, n (n = # of sites selected) 1. U4 2. U4 : n. U4	Y	Sites value. List of test sites to disable. Zero-length list indicates all sites. L, n (n = number of sites selected.) 1. ID of 1st site 2. ID of 2nd site … n. ID of nth site
ENABLE-SITE	RMCD	"ENABLE-SITE"	Y	
	CPNAME$_1$	"ENABLESITELIST"	Y	ENABLESITELIST parameter.
	CPVAL$_1$	L, n 1. U4 2. U4 : n. U4	Y	Sites value. List of sites to enable. Zero-length list indicates all sites.
PAUSE	RCMD	"PAUSE"	Y	
PP-SELECT	RCMD	"PP-SELECT"	Y	
	CPNAME$_1$	"PPID"	Y	PROCESSPROGRAMID parameter.
	CPVAL$_1$	A[80]	Y	The ID of the program to be loaded.
	CPNAME$_2$	"DEFINEDSITELIST"	N	DEFINEDSITELIST parameter.
	CPVAL$_2$	U4	N	DefinedSiteList value. Establish the initial default set of test-sites.
	CPNAME$_3$	"PROCESSPARAMETER"	N	PROCESSPARAMETER parameter.
	CPVAL$_3$	A[1..40]	N	ProcessParameter value. This is an optional parameter that has implementation specific meanings.
RESUME	RCMD	"RESUME"	Y	
	CPNAME$_3$	"PROCESSPARAMETER"	N	PROCESSPARAMETER parameter.
	CPVAL$_3$	A[1..40]	N	ProcessParameter value. This is an optional parameter that has implementation specific meanings.
START	RCMD	"START"	Y	
	CPNAME$_1$	"SELECTEDSITELIST"	N	SELECTEDSITELIST parameter.

Service	*SECS-II Field*	*Values*	*Req.*	*Description*
	CPVAL$_1$	L, n 1. U4 2. U4 : n. U4	N	Provides single execution override of test-sites enabled/disabled by remote commands ENABLE-SITE and DISABLE-SITE.
	CPNAME$_2$	"DATALOGDATASETS"	N	DATALOGDATASETS parameter.
	CPVAL$_2$	L, n (n - # of datasets) 1. L, 2 1. A[80] 2. List, p 1. U4 … p. U4 n. L, 2 1. A[80] 2. L, q 1. U4 … q. U4	N	Datalog output can be directed to separate datasets on a per-test-site basis. L, n (n - # of datasets) 1. L, 2 1. Dataset Name$_1$ 2. L, p (Zero-length list = all test-sites) 1. SiteID$_{11}$ … p. SiteID$_{1p}$ n. L, 2 1. Dataset Name$_n$ 2. L, q (Zero-length list = all test-sites) 1. SiteID$_{n1}$ … q. SiteID$_{nq}$
	CPNAME$_3$	"DATALOGBYTESTTYPE"	N	DATALOGBYTESTTYPE parameter.
	CPVAL$_3$	L, b 1. L, 3 1. U4 2. U4 3. U4 … b. L, 3 1. U4 2. U4 3. U4	N	Specifies datalog options by general TestType. L, b (b = # of (TestType, TestStatus) pairs being assigned reports.) 1. L, 3 1. TestType$_1$ 2. TestStatus$_1$ 3. ReportID$_1$ (If omitted, the default report is assigned.) … b. L, 3 1. TestType$_b$ 2. TestStatus$_b$ 3. ReportID$_b$
	CPNAME$_4$	"DATALOGBYTESTID"	N	