CIM300

Developer's Guide

Date: October 30, 2013

Table of Contents

1. General Information

All packages are COM objects and are useable from any language supporting COM.

Factory host emulation may be done with programs using SECSConnect or by using TESTConnect.

1.1 EPJ Files

Each package has an EPJ file template that is meant to be copied into the EPJ file for the equipment. From there you can customize the GEM IDs and default values. DO NOT change the names. The Wellknown name concept of CIMConnect is not used by CIM300.

Some packages require EPJ file customization to match the equipment configuration. This customization will be discussed in the Tutorials.

1.2 Equipment Manual Templates

Each package has a MS Word document that is the equipment manual template for that package. They are meant to be copied into the equipment GEM manual template provided with CIMConnect as new sections. Any changes to ID numbers made to the EPJ files will need to be made in the corresponding template.

1.3 Use of CIM300 Data Items

There are cases where the equipment application must use the events and data defined in the CIM300 EPJ files. In these cases, be sure to update the data and trigger the events using the CIMConnect SetValuesTriggerEvent API to ensure the integrity of the data with the event. This will prevent CIM300 from overwriting the applications values in the event report.

1.4 CIM300 Callback Implementation

Threading is an issue with callbacks. Some callbacks are the result of a host SECS message and others are from API calls within the equipment application.

Callbacks execute in an RPC thread if they are the result of a SECS message. It is not safe to update GUI components in these callbacks. These callbacks block the message processing in CIMConnect. Calling into CIM300 or CIMConnect API from within the callback may create a deadlock situation. Callbacks resulting from equipment application API calls execute in the application thread.

In general, it is not possible to know ahead of time which callbacks will execute in which thread. Callbacks should return as soon as possible and, where possible, process the callback information asynchronously.

If a callback is requesting the application to return data, then the callback must block execution and return a result. In these cases, try to be efficient and return as quickly as possible.

1.5 GEM Control State

Each CIM300 package object has a property called AllowRemoteControl which needs to be set based on the GEM Control State. If the Control state is Online-Remote, then AllowRemoteControl should be TRUE. Otherwise it should be FALSE to prevent the factory host from controlling the equipment using 300mm services.

1.6 Support

When contacting Cimetrix support for help with a CIM300 issue, send the following to support@cimetrix.com

- A copy of CIM300 logging output
- CIMConnect diagnostics file
- SupportTool output html file
    - Start->All Programs->Cimetrix->Tools->SupportTool
    - <installation folder>/bin/SendToCimetrix_XXXXXXXX.html
- If possible, a sample program that demonstrates the problem behavior

- A description of:
  a. The actual behavior of CIMConnect/CIM300
  b. The expected behavior of CIMConnect/CIM300
  c. How the problem was detected
  d. Affecting production in a fab?
  e. Affecting tool acceptance at a fab?

2. General Tutorials

2.1 Equipment Application Startup

Equipment applications should initialize following the procedure below:

1. Disable communications in CIMConnect.
2. Initialize the Cimetrix products in this order: CIMConnect, CIMFoundation, CIM90, CIM87, CIM40, CIM94, CIM116, CIM148 and CIM157.
3. Synchronize CIM300 with the equipment hardware.
4. Provide current values for all application-handled Status Variables.
5. Get the current values for all application-handled Equipment Constants.
6. Enable host communication if configured to do so in CIMConnect after CIM300 initialization and equipment initialization is complete to avoid invalid SECS-II message errors.

2.2 Equipment Application Shutdown

When the equipment application is shutdown, follow the procedure below to shut down Cimetrix products:

1. Disable host communication in CIMConnect by calling ICxClientApplication::DisableComm.
2. Stop all CIMConnect, CIM300 ActiveX controls if they are used in the GUI. Each has a “Stop” method.
3. Call Shutdown on each CIM300 module in the reverse order of startup: CIM157, CIM148, CIM116, CIM94, CIM40, CIM87, CIM90 and CIMFoundation.
4. Shutdown CIMConnect by calling ICxEMService::StopService.

2.3 CIM300 Diagnostic Logging

CIM300 has copious logging available for diagnostics. Each CIM300 package object has a property that determines the logging for that package. All CIM300 logging passes through CIMFoundation where the logging may be routed to CIMConnect to be saved in NSL files and/or to an application callback to be processed by the equipment control application.

The callback level logs the registration and unregistration of callback interfaces as well as when the interface is called. In addition, the result of the callback is logged whether that is a successful return, the callback throws an exception, or the callback object no longer exists. This can also be useful for profiling callbacks. The callback to perform the logging function is not logged.

The entry/exit level logs all API calls and many internal functions in CIM300 when they are entered and when they leave. This is useful for profiling and deadlock detection.

The error level logs all API calls that generate errors as they process requests from the application or the host. If an API returns a failure, details about that failure are logged.

CIMFoundation queues logged messages for asynchronous processing so as to not block CIM300 execution. The queue is processed on a background thread that will call the application callback and /or CIMConnect.

2.3.1 Package Logging Levels

Each CIM300 package object has a property that determines the logging level for that package. In this manner, logging can be tailored to the problem being diagnosed. In addition, this allows logging to be tuned for performance. Changing the logging property immediately affects logging for that package, no restart is required. This allows you to change logging in a running system, if needed, without a software restart.

The logging levels are explained in the table below. They may be added together to enable multiple levels of logging.

Level	Description
c3llNone	Logging is turned off

c3llError	Error messages. This can be useful for diagnosing why a CIM300 API failed.
c3llMessages	SECS-II messages.
c3llCallbacks	Callbacks to the application. This is useful to determine how long application callbacks take, if they threw exceptions or become unregistered.
c3llEntryExit	CIM300 API entry/exit logging. This is used to profile the CIM300 API and to look for deadlocks.
c3llDebug	Information logs. Useful for difficult problems and deadlocks.
c3llAll	All logging is enabled

2.3.2 Logging Callback

CIMFoundation supports a callback interface, ICxLogCallback, which may be implemented by the equipment control application and registered with the abstraction layer. If this is done, the LogCallback method will be called for each message to pass logs to the equipment application for processing.

2.3.3 Logging to CIMConnect

CIMFoundation has a property, LogToCIMConnect. When this property is VARIANT_TRUE, then the abstraction layer will pass the enabled log level messages on to CIMConnect. In CIMConnect, it will additionally be filtered according to the logging enabled in CIMConnect and written to an NSL (CIMConnect logging) file.

3. CIMFoundation

CIMFoundation includes the abstraction layer for CIMConnect as well as the CIM300 E39 package - CCAbsLayer.dll and CxE39OSS.dll.

The E39 package handles Stream 14 Functions 1, 3, 5, 7, 19 and 23 for all objects automatically. Substrate and Control Jobs support Functions 9 and 11, Create and Delete services.

3.1 Initialization

CIMFoundation must be initialized before any of the other packages. To initialize CIMFoundation, follow the procedure below:

1. Call ICCAlObject::Init(). The connection number should be the CIMConnect connection number the factory host will use for 300mm SECS messaging. It is usually 1, corresponding to [CONNECTION1] in the epj file. Only this connection will accept 300mm SECS stream and functions such as S3F17.
2. Call ICxE39::Init. The name of the equipment E39 object at the top of the object hierarchy must be passed in. This is commonly “top:top”. Also the interface to the abstraction layer must be passed in, that is why this step is second.

3.2 Custom E39 Objects

It is possible for the equipment application to implement custom E39 objects such as load ports. To do this, follow the procedure below:

1. Add a new COM object to your equipment application that will be the class for the new E39 object.
2. Aggregate or inherit CxE39Obj to provide the default E39 object behavior.
    a. In the class constructor, call the base class Init method with the ObjSpec for the object. For example Init(“LoadPort:LP1”);. This sets the name and type of the E39 object.
3. Implement ICxE39CB to handle SetAttr requests and enforce readonly permissions.
    a. Return E_NOTIMPL for most methods.
    b. Return S_OK for GetAttr.
    c. Return S_OK for SetAttr. Set ErrCode to 0 if it is okay for the host to change the value of the attribute. Set ErrCode and ErrText to a specific error code and message to deny the setting of the attribute.
4. In the new class, call the base class SetAttr API to change the E39 object attributes to the values that are visible to the host.
5. In the new class, call Add on the E39 package object to add the new E39 object into the hierarchy and make it available to the host.

3.3 Custom Attributes

The equipment application may add custom E39 attributes to any E39 object by simply calling SetAttr on the object. It would need to do this for each attribute every time its value changes. These attributes will be collectable by the factory host.

3.4 E39 Informational Callbacks

The E39 package has a callback interface to provide change information to the application, ICxE39InfoCB2. When registered with the E39 package object, it is used by E39 to notify the application when E39 objects are created or deleted and when an object’s (any E39 object) attribute value changes. CIMFoundation queues these notifications for asynchronous processing so as to not block CIM300 execution. The queue is processed on a background thread that will call the application callback. Use of ICxE39InfoCB2 callback is preferred over ICxE39InfoCB as the ICxE39InfoCB is synchronous and does block execution of CIM300. This callback should be used sparingly as it will be called often and will impact overall performance of the application.

3.5 Shutdown

When the equipment application is shutdown, it should call Shutdown() on the CxE39Obj and CCAlObject to shut down the E39 package and abstraction layer. This releases any callbacks that were registered and dependent interfaces the packages may have.

4. CIM90

CIM90 provides objects to help implement the E90 standard.

4.1 Overview

4.1.1 Substrate Object

CIM90 provides the E39 compliant Substrate object. A substrate has both a state indicating where it is located and a state indicating the progress of processing. These states are represented by the SUBSTRATE TRANSPORT and the SUBSTRATE PROCESSING states, which are concurrent substrates of the SUBSTRATE State Model. CIM Foundation is the prerequisite of CIM90 implementation.

4.1.2 Substrate Location Object

CIM90 provides the E39 compliant SubstrateLocation object. SubstrateLocations are used to model any place on the equipment a substrate may be placed. Examples are robot arms, aligner chucks, load lock slots and process modules. The SubstrateLocation has a state indicating if it is occupied or unoccupied by a substrate.

4.1.3 Batch Object

CIM90 provides an object representing a collection of wafers known as a batch. A set of wafers moved as a unit is referred to as a batch.

4.1.4 BatchLocation Object

CIM90 provides an object for defining places in the equipment where sets of wafers may be placed. The batch location has the occupied and unoccupied states.

4.2 EPJ File

The E90 epj file should be copied into the epj file for the equipment. Once that is done, there are adjustments that must be made to match the equipment. The value of the E90ComplianceLevel parameter may need to be set to match the E90 standards compliance level for the version of E90 the equipment is to support.

For single wafer equipment, each substrate location within the equipment must have a set of subscripted SVs. They are the SubstLocState_i, SubstrLocSubstrID_i, SubstLocID_i, SubstLocDisableEvents_i parameters where ‘i’ is the number of the location. Each static (robot arm, aligner, etc.) and dynamic (carrier) location must have a set of these parameters.

For a batch tool, each batch location must have a similar set of parameters: BatchLocDisableEvents_i, BatchLocID_i, BatchLocState_i, BatchSubstIDMap_i.

4.3 Initialization

Follow the procedure below to initialize the E90 package:

1. Call ICxE90ST::Initialize to connect CIM90 to CIMFoundation.
2. (Obsolete) Define the static substrate locations in the equipment by calling ICxE90ST::AddSubstrateLocation for each location. The order of calls to AddSubstrateLocation determines that location’s index and therefore which set of subscripted variables in the EPJ file are used for that location.

   (Preferred) Define the static substrate locations in the equipment by calling ICxE90ST::AddSubstrateLocation2 for each location. The subscriptIndex parameter for AddSubstrateLocation2 determines that location’s index and therefore which set of subscripted variables in the EPJ file are used for that location. This provides more control to the application over the subscript index for the substrate location.
3. (Obsolete) Define the batch locations by calling ICxE90ST::AddSubsrateBatchLocation. The order of calls to AddSubstrateBatchLocation determines that location’s index and therefore which set of subscripted variables in the EPJ files are used for that location.

(Preferred) Define the batch locations by calling ICxE90ST::AddSubsrateBatchLocation2. The subscriptIndex parameter for AddSubstrateBatchLocation2 determines that location's index and therefore which set of subscripted variables in the EPJ files are used for that location. This provides more control to the application over the subscript index for the substrate's batch location.

4.4 Programming

4.4.1 Substrate Creation and Removal

Normally, the Substrate objects will be created by CIM87 when the Carrier slotmap has been verified. SetCIM90Interface API on CIM87 must be called so that CIM90 automatically creates/deletes substrates on carrier arrival/departure. In the event the substrates are introduced to the equipment in some other manner they may be created using the (now obsolete) RegisterSubstrate or RegisterSubstrate2 APIs. When needed, the preferred method is to use the new RegisterSubstrate3 or RegisterSubstrate4 APIs, allowing specifying the substrate's subscriptIndex. To remove the substrate if it leaves the equipment other than in a carrier, the RemoveSubstrate API is used.

4.4.2 Persistence

Substrate objects may be persisted. This is often wanted to save the state of the equipment in case of power failure or equipment shutdown to be able to restore that state when the software restarts. Typically, the tool software has extra questionnaires when it recovers from the error condition or hard shutdown as substrates may be removed from the equipment manually. However, the application may use the ICxE39Obj SaveObjState and RestoreObjState methods on the E39 package object to manage persistence.

4.4.3 Substrate Movement and Processing Changes

Call ChangeSubstrateState or ChangeSubstrateState2 whenever a substrate is moved or its processing state changes. This method will update data and trigger state machine events for you. It will generate timestamps automatically for the substrate history. If you wish to specify the timestamps yourself, use ChangeSubstateStateAndHistory or ChangeSubstrateStateAndHistory2.

4.4.4 Batch Operations

4.4.4.1 Add Batch Locations

Call AddSubstrateBatchLocation or AddSubstrateBatchLocation2 API (preferred) to define all batch locations such as load lock, drying station, process chamber or any location where the bacth will be placed after the substrates are moved from the carrier. AddSubstrateBatchLocation2 API can be used to choose a specific subscripted index, otherwise, CIM90 automatically grabs the available subscripted index and updates BatchLocDisableEvents_i, BatchLocID_i, BatchLocState_i, BatchSubstIDMap_i.

4.4.4.2 Create Batch of Substrates

Call ICxE90ST::CreateBatch to create a batch with an id and location. To add all substrates from the specified carrier to the specified batch, call ICxE90ST::AddCarrierToBatch or ICxE90ST::AddCarrierToBatch2 (preferred).

Call ICxE90ST::AddSubstrateToBatch to add the specified substrate to a batch.

4.4.4.3 Batch Movement

Call ChangeBatchState anytime a batch moves to new location or changes its processing state. All DVs, events, timestamp are managed by CIM90. ChangeBatchStateAndHistory is used in case you are interested to provide the timestamp.

4.4.4.4 Remove Substrate / Batch locations

Call ICxE90ST::RemoveSubstrateFromBatch to remove a single substrate from the batch. To remove all substrates added by AddCarrierToBatch2, call ICxE90ST::RemoveCarrierFromBatch.

Call ICxE90ST::RemoveSubstrateBatchLocation to remove the batch location. You can call ICxE90ST::DestroyBatch to delete the batch. This API does not require the substrate be removed first.

4.4.5 Substrate ID Verification

Substrate ID Verification is only used if the equipment has a substrate ID reader. This is usually part of the aligner. Use the NotifySubstrateRead API after the substrate ID has been read to notify the host.

4.4.6 Useful Functions

- ICxE87CMS::GetSubstrateFromSource - ICxSubstrate from the source carrier and slot number
- ICxE90ST::GetSubstrate - ICxSubstrate object from the substrate ID
- ICxSubstrate::objectID - Get the Substrate ID
- ICxSubstrate::substrateSource - Format “<CarrierID>.<slot number>”
- ICxSubstrate::substrateDestination - Format “<CarrierID>.<slot number>”
- ICxE90SubstrateCB::SubstrateStateChange

4.5 Callbacks

CIM90 has a callback interface the application may implement to receive substrate commands for Substrate ID verification (E90CommandCallback) or substrate state changes (SubstrateStateChange). To use these, the equipment application would need a COM object implementing the ICxE90SubstrateCB interface. Then the equipment application would create an instance of that object and register it with the CIM90 package object during initialization using ICxE90ST::RegisterCallback.

E90CommandCallback is used to notify the application if the host verifies or rejects the substrate ID when ProceedWithSubstrate and CancelSubstrate APIs are called. The callback will be called for both APIs.

SubstrateStateChange is called when a substrate state changes. This is a notification callback, the equipment software does not reject or accept it.

4.6 Shutdown

When the equipment application is shutting down, call Shutdown() on the CxE90ST package object to release any registered callbacks and dependent interfaces.

4.7 FAQ

1. Q: Does AddSubstrateLocation() create substrate using the non-empty substrateID that is passed in?

   A: AddSubstrateLocation does not create the substrate, RegisterSubstrate() does. If either substrate location or substrate does not exist, RegisterSubstrate() will create either substrate location object or substrate object for you.

2. Q: Do I have to create substrates or does CIM300 do it for me?

   A: As long as you call the SetCIM90Interface() API on the CIM87 package, CIM87 will create and delete all the substrates that arrive in carriers. The substrates are created as soon as the slotmap is verified and are deleted when you call CarrierDepartedPort().

3. Q: How do I add customer attribute to substrate or substrate location object?

   A: Customer attribute can be added to these objects using ICxE39Obj::SetAttr() method as described in Adding and Setting Custom Attributes. These customer attributes are read/write, so the host can change their values by sending S14S3.

4. Q: What is the naming convention used for substrate locations?

   A: If you set the CIM90 Interface in CIM87, CIM87 will create substrates and substrate locations after the slot map is verified OK. These carrier substrate location objects are in the form of <carrierID>.01

through <carrierID>.25 and can be retrieved using the ICxE90ST::GetSubstrateLocation() API. If the substrate locations are created using the ICxE90ST::AddSubstrateBatchLocation() API, the object ID's of these CxSubstrateLocation objects are in the form of <locationGroupID>.01 through <locationGroupID>.numOfLocations.

5. CIM87

5.1 EPJ File

The E87 epj file should be copied into the epj file for the equipment. Once that is done, there are adjustments that must be made to match the equipment. The value of the E87ComplianceLevel parameter may need to be set to match the E87 standards compliance level for the version of E87 the equipment is to support.

Each load port within the equipment must have a set of subscripted SVs. They are the PortStateInfo_i, PortAssociationState_i, PortTransferState_i, PortID_i, AccessMode_i, LoadPortReservationSate_i and optionally ByPassReadID_i parameters where ‘i’ is the portID. In addition, most load ports have two carrier locations, the load/unload location and the docked location. These locations also require CarrierID_i and LocationID_i parameters (2 each).

In addition, each carrier location that is not a load port (such as internal buffer locations) must have a set of parameters: CarrierID_i, LocationID_i.

(Obsolete) If methods without the subscriptIndex are used, the order that load ports and carrier locations are created in CIM87 determines which indexes are used for which locations.

(Preferred) If methods with the subscriptIndex are used, the subscriptIndex will be used to determine the index for the location.

5.2 Initialization

Call ICxE87CMS::Init to connect CIM87 to E39 and the abstraction layer. In addition, call ICxE87CMS::SetCIM90Interface so that CIM87 can create substrates and substrate locations for carriers.

Register for callbacks using the ICxE87CMS::RegisterCallback API.

(Obsolete) Model each load port by calling ICxE87CMS::Create2LocationLoadPort. For internal buffer equipment, call AddCarrierLocation and CreatePartition to model the internal buffer(s).

(Preferred) Model each load port by calling ICxE87CMS::Create2LocationLoadPort2. For internal buffer equipment, call AddCarrierLocation2 and CreatePartition2 to model the internal buffer(s). This provides more control to the application over the subscript index for the load ports.

5.3 Programming

5.3.1 Carrier Delivery

The API used during carrier delivery and the sequence is described here.

1. When the beginning of delivery is known, call ICxE87CMS::TransferStart to change the load port state from *Ready To Load* to *Transfer Blocked*. For AMHS deliveries, this is usually done on an E84 signal. For manual deliveries, this is usually done on carrier placement.
2. After the carrier has been placed, call the ICxE87CMS::CarrierPlaced API.
3. This is usually a good time to trigger the GEM MaterialArrived collection event.
4. After the carrier has been clamped, trigger the CarrierClamped collection event with the PortID, CarrierID and LocationID DVs using SetValuesTriggerEvent.
5. After the ID has been read, determine if there is another Carrier being processed with the same ID as the new Carrier and, if so, trigger the DuplicateCarrierIDInProcess collection event with the CarrierID DV using SetValuesTriggerEvent.
6. Call ICxE87CMS::CarrierAtPort.
7. Once the ID has been verified and the CarrierIDVerified callback has been called, dock the FOUP, open it and scan the slot map.
8. After the FOUP has been docked, call ICxE87CMS::MoveCarrier to logically move the carrier from the load/unload position of the load port to the docked position of the load port.
9. Once the carrier has been opened, trigger the CarrierOpened collection event with the PortID, CarrierID and LocationID DVs using SetValuesTriggerEvent.

10. After the slot map has been scanned, call ICxE87CMS::VerifySlotMap.
11. If the slot map is verified, the SlotMapVerified callback will be called.

If the host rejects the FOUP at either ID or slot map verification, the CancelCarrier callback will be called by CIM87 to notify the equipment application that it should make the FOUP ready to unload.

5.3.2 Carrier Removal

The API used during carrier removal and the sequence is described here.

1. Once the carrier has been completed (the last wafer returned to the carrier) and the door closed, trigger the CarrierClosed collection event with the PortID, CarrierID and LocationID DVs using SetValuesTriggerEvent.
2. If the CarrierHold EC is set to Host, then wait to undock the carrier until the CarrierRelease callback is received. Otherwise undock the carrier.
3. Once the carrier is undocked, call ICxE87CMS::MoveCarrier to logically move the carrier from the docked position to the load/unload position. Also call ICxE87CMS::ReadyToUnload to change the load port state to *Ready To Unload*.
4. If the CarrierUnclamp EC is set to 0, unclamp the carrier as soon as it is undocked. Otherwise wait until the AMHS signal or the operator to press a button.
5. When the carrier has been unclamped, trigger the CarrierUnclamped collection event with the PortID, CarrierID and LocationID DVs using SetValuesTriggerEvent.
6. When the transfer begins, call ICxE87CMS::TransferStart to take the port to *Transfer Blocked*. For AMHS deliveries, this is usually done on an E84 signal, for manual deliveries this is usually done on when the operator presses a button on the load port or the GUI.
7. When the carrier has been removed, call ICxE87CMS::CarrierDepartedPort. This will destroy the CIM90 SubstrateLocation and Substrate objects associated with the carrier as well as the CIM87 carrier object. It will also change the load port to *Ready To Load*.
8. This is usually a good time to trigger the GEM MaterialRemoved collection event.

5.3.3 Accessing the Carrier

When the first substrate is pulled from the carrier, call ICxE87CMS::AccessCarrier to set the carrier accessing status to IN_ACCESS. Once the last substrate is put back in the carrier, call ICxE87CMS::AccessCarrier to mark the carrier either CARRIER_COMPLETE or CARRIER_STOPPED. Once the carrier has been marked COMPLETE or STOPPED, it may be undocked and made ready to unload.

5.3.4 Local Operation

Use the CancelCarrier and ProceedWithCarrier API to verify or cancel a carrier for both ID and slotmap verification from the equipment application.

5.3.5 Carrier ID Reader Availability

If the carrier ID reader on the equipment becomes unavailable due to hardware problems or removal, trigger the IDReaderUnavailable event with the PortID DV. If it becomes available again, trigger IDReaderAvailable. Use SetValuesTriggerEvent for this.

5.3.6 Load Port In and Out of Service

If a load port is taken out of service, then ChangeLPTransferState should be called. This will notify the host that the port has been taken out of service and should not be used. When the port is back in service, call ChangeLPTransferState again.

5.3.7 Load Port substrate indices

By default, when a substrate arrives on a load port, its subscript indices are assigned based on what indices are currently available. To override this behavior and have deterministic control over the values assigned, the base index and the number of substrates in a carrier can be set with the ChangeLPSubstrateIndexRange() API call. This has to be done after the load port is created.

5.3.8 CarrierReCreate

In order for the CarrierReCreate service to be accepted by CIM87, the load port must be *Ready To Unload*. Once the application receives a CarrierReCreate callback, it should follow a normal carrier delivery sequence starting at the CarrierAtPort step.

5.3.9 Persistence

ICxE87CMS::CreateCarrier may be used to restore a carrier object after a loss of power or other tool shutdown. In addition, the ICxE87CMS::put_TransferStatus may be used to initialize a load port to a specific state. CIM87 persists load port states across restarts, so this initialization may be needed after scanning the actual hardware for changes on startup.

5.3.10 Alarms

E87 has several alarms that must be managed by the equipment application. They are: AccessModeViolation, CarrierPresenceError, CarrierPlacementError, CarrierDockFailure, CarrierOpenFailure, CarrierRemovalError, PIOFailure, IBCarrierMoveFailure (internal buffer equipment only).

In addition, CIM87 will automatically set the following load port alarms: DuplicateCarrierID, CarrierVerificationFailure, AttemptToUseOutOfServiceLP and SlotMapVerificationFailed.

Some fabs require each load port to have unique alarms. The ICxE87Callback8::PortAlarmChanged callback may be helpful to synchronize those port-specific alarms with the alarms CIM87 automatically sets.

5.3.11 Internal Buffer Equipment

Model the movement of the carrier from the load port to a buffer location using one of the MoveCarrier, MoveCarrier2, or MoveCarrier3 APIs. Also pay attention to the CarrierIn, CarrierOut, CancelCarrierOut and CancelAllCarrierOut callbacks.

5.3.12 Tag Read/Write

The host may want to read or write carrier tag data. The equipment application will be notified of these requests through the CarrierTagReadData and CarrierTagWriteData callbacks. CarrierTagReadData is a synchronous call and the data is expected to be returned from the callback.

5.3.13 Manual transfer in AUTO mode

5.3.13.1 Manual Carrier placement

Preconditions: LPTSM=READY TO LOAD, LPAMS=AUTO, ES=ON, HO_AVBL=ON

1. Operator places Carrier.
2. Call CIM87 TransferStart. CIM87 triggers the LPTSM SCT #6 (E87ReadyToLoad2TransferBlocked) event.
3. Trigger MaterialReceived event.
4. Set AccessModeViolation alarm. CIMConnect sends S5F1 and triggers the AccessModeViolationSet event.
5. Option 1: Operator selects "Continue"
    a. Clear AccessModeViolation alarm. CIMConnect sends S5F1 and triggers the AccessModeViolationClear event.
    b. Call CIM87 CarrierAtPort.
    c. Continue operations as normal.
6. Option 2: Operator selects "Cancel"

   a. Operator removes Carrier.
   b. Clear the AccessModeViolation alarm. CIMConnect sends S5F1 and triggers the AccessModeViolationClear event.
   c. Call CIM87 ChangeLPTransferState with LP_READY_TO_LOAD. CIM87 triggers the LPTSM SCT #8 (E87TransferBlocked2ReadytoLoad) event.

5.3.13.2 Manual Carrier removal

Preconditions: LPTSM=READY TO UNLOAD, LPAMS=AUTO, ES=ON, HO_AVBL=ON

1. Operator removes Carrier.
2. Call CIM87 TransferStart. CIM87 triggers the LPTSM SCT #7 (E87ReadyToUnload2TransferBlocked) event.
3. Set the AccessModeViolation alarm. CIMConnect sends S5F1 and triggers the AccessModeViolationSet event.
4. Trigger the MaterialRemoved event.
5. Call CIM87 CarrierDepartedPort. CIM87 triggers the LPTSM SCT #8 (E87TransferBlocked2ReadytoLoad) event.
6. Clear the AccessModeViolation alarm. CIMConnect sends S5F1 and triggers the AccessModeViolationClear event.

5.4 Callbacks

There are many callbacks for CIM87 just as there are many services in E87.

- To reject an E87 Service, return E_NOTIMPL in the associated callback.
    - In .NET, that means throw a COM exception
    - In other words, a fixed buffer tool should reject CarrierIn, CarrierOut, CancelAllCarrierOut and CancelCarrierOut.
- To support an E87 Service, return S_OK.
    - In .NET, this is the default.
- Use the newer callbacks for appropriate services.
    - For example, use ICxE87Callback5::CancelCarrier2 instead of ICxE87Callback::CancelCarrier.

5.5 Shutdown

When the equipment application is shutting down, call Shutdown() on the CxE87CMS package object to release any registered callbacks and dependent interfaces.

6. CIM40

CIM40 provides an E39 compliant ProcessJob object. Process jobs relate a processing recipe to a set of material. Process jobs also have a state machine. The process job is a transient entity. It is created on request of the supervisor, executes and then is deleted by the processing resource. The job usually spans the time period from shortly before material is physically delivered to the processing resource, through the processing and until shortly after material is taken away.

6.1 EPJ File

The E40 epj file should be copied into the epj file for the equipment. The E40ComplianceLevel EC may need to be adjusted for the version of the SEMI E40 standard you wish to support.

6.2 Initialization

After creating the E40 package object, call ICxE40PJM::Initialize2 to link CIM40 to E39, the abstraction layer and CIM87. This is also used to set the PJ queue size. At least 25 jobs per load port is a recommended starting point. The Initialize2 API also requires a pointer to an equipment application CIM40 callback object so that CIM40 can request the application to perform process job related activities. If Initialize2 API is not used then CIM87 wouldn’t delete the process job when the material is removed.

6.3 Programming

6.3.1 Local Jobs

Use the ICxE40PJM::PRJobCreate or PRJobCreateEnh API to create jobs from the equipment application. This is usually followed by creating a Control Job containing the newly created process jobs.

6.3.2 PRCommandCallback

The PRCommandCallback callback function is used by CIM40 to notify the equipment application that it should do something with a process job. The suggested behavior in each callback is detailed in the sections below.

6.3.2.1 Create

Verify the process job parameters such as the recipe and any recipe parameters specified. If something in the job definition is incorrect, specify the error(s) in the prStatus argument of the callback and return E_FAIL.

6.3.2.2 Setup

This callback is used to notify the application of the next process job to be run. The application should take this information and asynchronously prepare the equipment to run the process job. Do not perform these activities in the callback as that blocks CIM40. When the setup activities have completed, the equipment application should call the ICxE40PJM::PRJobStartProcess API to let CIM40 know the job setup has completed for the job.

6.3.2.3 Start

CIM40 uses this callback to notify the equipment application that it should begin executing the process job.

6.3.2.4 Abort, Stop, Pause, Resume

CIM40 uses these commands in the callback to tell the equipment application to perform the corresponding action on the process job. Abort and Stop lead to the process job completing and the section on ProcessJob Completion should be followed as soon as all substrates have been returned to carriers.

6.3.3 ProcessJob Completion

Process jobs are started automatically from CIM94. When actual processing of the process job completes, call ICxE40PJM::PRJobProcessComplete. This notifies the host that the job has finished processing. This is normally done when processing of the last substrate in the job has finished, but before that substrate is moved back to the carrier. Once the last substrate in the job is back in the carrier, call ICxE40PJM::PRJobComplete to compete and terminate the ProcessJob.

6.3.4 ProcessJob Management

Use the PRStop, PRAbort, PRPause,and PRResume API to control process job execution. These API calls will result in PRCommandCallback callbacks into the equipment application requesting those activities. CIM40 will take care of state machine transitions and event reporting.

6.4 Callbacks

Add a COM object to the equipment application and implement ICxE40PJMCalback and ICxE40PJMCallback2. Use ICxE40PJMCallback2::PRCommandCallback for process job commands instead of ICxE40PJMCallback::PRCommandCallback.

6.5 Shutdown

When the equipment application is shutdown, it should call Shutdown() on the CxE40PJM package object. This releases any callbacks that were registered and dependent interfaces the packages may have.

7. CIM94

E94 defines Control Jobs as a unit of work on equipment for one or more carriers. The work is described by a set of one or more process jobs to be applied to the material contained in the carriers. CIM94 provides the E39 compliant ControlJob object and API to create and manipulate them. CIM94 must be used with CIM40.

7.1 EPJ File

The E94 epj file should be copied into the epj file for the equipment. Once that is done, there are adjustments that must be made to match the equipment.

7.2 Initialization

Call ICxE94CJM::Init and ICxE94CJM::SetE87 to initialize CIM94. In addition, register the equipment application callback using one of: ICxE94CJM::RegisterCallback, ICxE94CJM::RegisterCallback2, or ICxE94CJM::RegisterCallback3. Use of callback 3 is recommended as it has superseded callback 2.

7.3 Programming

List and Arrival ProcessOrderMgmt are handled automatically.

7.3.1 Local Jobs

Use ICxE94CJM::CreateControlJob to create a job specified from the equipment application. Use CJStart, CJStop, CJPause, CJResume, or CJAbort to manage jobs from the equipment application.

7.3.2 Running Jobs

Be sure to call ICxE94CJM::ProcessJobComplete when a process job has completed so CIM94 will start the next available process job or control job.

7.3.3 Material Redirection

The equipment may optionally support Material Redirection Mode by allowing the MtrlOutSpec attribute to be non-empty (i.e., if it contains at least one SourceMap and DestinationMap pair).

CIM300 does not implement this feature directly; the equipment application needs to implement the following to support it.

7.3.3.1 Equipment does not support Material Redirection

E94-0308 section 8.4.5.3.2 reads: “If the equipment does not support Material Redirection Mode, it shall reject any request to set a value for MtrlOutSpec.”

In order to reject Control Job creation, the application needs to implement the ICxE94ControlJobCB3 callback interface and register it using the ICxE94CJM::RegisterCallback3 method. When the host tries to create a new Control Job, the ICxE94ControlJobCB3::CJCommandCallback is called with cjCmd parameter set to cjCommandCreate. The application needs to get the value of “MtrlOutSpec” attribute from the pJobObj object (using ICxE39Obj::GetAttr method) and, if not empty, set prStatus value to reject the create request.

In order to reject S14F3 SetAttr request from the host, the application needs to implement the ICxE94ControlJobCB4 interface and register it using the ICxE94CJM::RegisterCallback4 (before ControlJob creation) or ICxE94ControlJob::RegisterCallback4 (for each ControlJob as they are created). When the host tries to set an attribute, the ICxE94ControlJobCB4::SetAttrChangeRequest method will be called. In the callback handler, compare the value of the attrName parameter with “MtrlOutSpec”. If the comparison is true, set the errCode and errText parameter values. A non-zero errCode value rejects the SetAttr request from the host.

7.3.3.2 Equipment supports Material Redirection

When starting substrate processing, before removing it from the carrier, the application needs to get the “MtrlOutSpec” attribute value from the Process Job’s parent Control Job. If MtrlOutSpec is not empty and it contains information about this particular substrate, set Substrate’s “SubstDestination” attribute to the ID of the target Substrate Location.

After substrate processing is complete, the application needs to read the Substrate object’s “SubstDestination” attribute move the substrate into that location. If the destination is occupied or inaccessible, the equipment shall set the SubstrateDestinationNotAccessible alarm. The equipment shall clear the alarm when a DestinationMap referring to an accessible destination has been provided by the host or operator intervention.

If the equipment supports incomplete MtrlOutSpecs (See SEMI E94-0308 standard, section 8.4.5.3.6), your application should allow updates of the DestinationMaps specified in the MtrlOutSpec attribute before or after the ControlJob enters the EXECUTING state. Requests in the EXECUTING state shall only be accepted if the SourceMaps contained in the MtrlOutSpec remain unchanged and material whose destination location is changed by the update has not yet been placed into a destination carrier and is not currently moving into a destination carrier.

In order to accept or reject S14F3 SetAttr request from the host, the application needs to implement the ICxE94ControlJobCB4 interface and register it using the ICxE94CJM::RegisterCallback4 (before ControlJob creation) or ICxE94ControlJob::RegisterCallback4 (for each ControlJob as they are created). When the host tries to set an attribute, the ICxE94ControlJobCB4::SetAttrChangeRequest method will be called. In the callback handler, compare the value of the attrName parameter with “MtrlOutSpec”. If the comparison is true, set the errCode and errText parameter appropriately. Your application should also update the Substrate object’s “SubstDestination” attribute if it has been changed.

If the equipment supports incomplete MtrlOutSpecs, after substrate processing is complete, if the DestinationMap is empty, the equipment shall set the SubstrateDestinationUnknown alarm. The equipment shall remain capable of accepting updates to the MtrlOutSpec as described above. The equipment shall automatically clear the SubstrateDefinitionUnknown alarm when the material has successfully moved to its destination location.

7.3.4 Optimize Job Order

If the equipment application will support the Optimize ProcessOrderMgmt option it will need to implement the ICxE94ControlJobCB::GetNextPrJobID to inform CIM94 of the next process job that should be run for a given control job.

7.3.5 Concurrent Jobs

To run jobs concurrently, call either ICxE94CJM::StartNextProcessJob to start the next process job within a control job or ICxE94CJM::StartNextControlJob to start the next control job.

7.3.6 Chaining Jobs

To maintain throughput, it is usually desirable to chain jobs. In chained jobs, the next job is started when the last wafer of the job is picked from the carrier. With this, there will be multiple jobs executing simultaneously. This is done by calling ICxE94CJM::StartNextControlJob or StartNextProcessJob.

7.4 Callbacks

Add a COM object to the equipment application and implement ICxE94ControlJobCB, ICxE94ControlJobCB3 and ICxE94ControlJobCB4. The CJCommandCallback will be called by CIM94 to notify the equipment application when a control job command has been issued. The equipment application may use the callback to reject the command if it is not possible given the state of the equipment. The methods on ICxE94ControlJobCB of interest are CJCreate and CJStateChange. The ICxE94ControlJobCB4 interface provides a method called SetAttrChangeRequest which is used to support/reject material redirection (see SEMI E94-0308, section 8.4.5.3 and 7.3.3).

7.5 Shutdown

When the equipment application is shutting down, call Shutdown() on the CxE94CJM package object to release any registered callbacks and dependent interfaces.

8. CIM116

E116 Equipment Performance Tracking standard requires E39 EPTTracker objects for each module of the equipment that affects processing or throughput. The EPTTracker object has an associated state model.

The EPT state model is intended to capture the different states of the equipment and its EPT modules from an operational point of view:

- EPT module busy executing a task.
- EPT module blocked from executing a task.
- EPT module idle.

8.1 EPJ File

The E116 epj file should be copied into the epj file for the equipment. Once that is done, there are adjustments that must be made to match the equipment. There are a set of status variables required for each EPT Module on the equipment including: EPTElementName_i, EPTState_i, PreviousEPTState_i, EPTStateTime_i, BlockedReason_i, BlockedReasonText_i, SubstCount_i, SubstCountReset_i, DisableEventOnTransition_i, TaskName_i, TaskType_i, PreviousTaskName_i and PreviousTaskType_i. Also, each module requires an EPTStateChange_i event. There needs to be one set of these for each EPT module on the equipment.

(Obsolete) The ‘i’ is an index number suffix that corresponds to the order in which the module was added to CIM116 during initialization.

(Preferred) The ‘i’ is an index number suffix that corresponds to the subscriptIndex passed to one of the add module methods during initialization.

8.2 Initialization

Follow the procedure below to initialize CIM116:

1. Call ICxE116Object::Initialize with the abstraction layer, E39 package object and the name of the equipment.
2. (Obsolete) For each wafer handling module (load locks, robots, aligner, etc.) call AddEFEMModule to create an EPT module to represent it.

   (Preferred) For each wafer handling module (load locks, robots, aligner, etc.) call AddEFEMModule2 to create an EPT module to represent it. This provides more control to the application over the subscript index for the module.
3. (Obsolete) For each process module, call AddProductionModule.

   (Preferred) For each process module, call AddProductionModule2. This provides more control to the application over the subscript index for the module.
4. (Obsolete) Note: the order of calls to AddEFEMModule and AddProductionModule determines the index used for the module status variables in the EPJ file.

   (Preferred) Note: the subscriptIndex parameter for AddEFEMModule2 and AddProductionModule2 determines the index used for the module status variables in the EPJ file.

8.3 Programming

Call ICxE116Object::Busy for each module when it begins executing a task. Call Idle when it completes a task. Call Blocked if the module becomes unable to perform any further tasks.

The IncrementSubstrateCounter, AddToSubstrateCounter and ResetSubstrateCounter APIs may be ignored as they are not required by the standards.

8.4 Callbacks

CIM116 has two callback interfaces that may be used. ICxEPTCallback3 has a method StateChange which may be used by the equipment application for logging EPT Module state changes and is called whenever any module changes state. ICxEPTCallback2 has a method IsEquipmentBlocked that may be called by CIM116 to ask the equipment application if the overall equipment is blocked from execution. ICxE116Callback has been deprecated and replaced by ICxEPTCallback3.

8.5 Shutdown

When the equipment application is shutting down, call Shutdown() on the CxE116Object package object to release any registered callbacks and dependent interfaces.

9. CIM148

E148 Time Synchronization standard requires a single E39 TS-Clock object instance for the equipment. CIM148 provides an implementation for this E39 object that handles S14F1 GetAttr Request and S14F3 SetAttr Request messages.

9.1 EPJ File

E148 standard does not require any GEM variables, collection events or alarms and so no EPJ file changes required.

9.2 Initialization

Follow the following procedure to initialize CIM148:

1. Create an instance of CCxTSObject object.
2. Call ICxTSObject Initialize method.
3. Optional: If you want to be able to reject requests from GEM Host to change R/W attribute (TimeServers and UseTimeSync) values, implement ICxTSObjectRequestCB callback interface and register it using ICxTSObject::RegisterRequestCallback.
4. Implement ICxTSObjectInfoCB callback and register it using ICxTSObject::RegisterInfoCallback method to receive notifications about TimeServers and UseTimeSync attribute value changes.
5. Query status from the NTP Client and update values of LastSyncTime, LastSyncTimeServer, Offset, Status, TimeServers, TimeSyncInterval and UseTimeSync properties of CCxTSObject.
6. Optional: Change value of CCxTSObject::ReportDateTimeInUTC property to “false” to report DateTime values in Local time format to the GEM Host. The default setting is to report time in UTC.
7. Add CCxTSObject object to E39 package object (the “top” E39 object) using ICxE39Obj::Add method.

9.3 Programming

When using CIM148, CIMConnect should be configured as follows:

1. Change AutoRejectS2F31 default value to “Bo 1” to reject requests from GEM Host to update system clock.
2. Remove ClockOffset variable(s) and change UpdateSystemClock default value to “Bo 1” in order for CIMConnect to always use real system clock when reporting time to the GEM Host.
3. Change ExtendedTimeFormat value to match the CIM148 ReportDateTimeInUTC setting.

CIM148 allows for two different, mutually exclusive ways of publishing NTP status data:

1. Preferred way, if NTP software has a way of notification when the system clock has been synchronized with the NTP server:
   a. Upon receiving notification from NTP client software about clock synchronization, update the following CCxTSObject properties: LastSyncTime, LastSyncTimeServer, Offset, Status and TimeSyncInterval.
2. Otherwise, if NTP status can only be queried:
   a. Implement ICxTSObjectGetAttrValueCB callback interface and register it using CCxTSObject::RegisterGetAttrValueCallback.
   b. CIM148 will call ICxTSObjectGetAttrValueCB::GetAttrValue method whenever GEM Host or an application requests E148 attribute values.
   c. In your implementation of ICxTSObjectGetAttrValueCB::GetAttrValue method, query NTP data for value specified in the AttrName parameter and return the value in the AttrValue parameter.

Update NTP Client settings when GEM Host changes TimeServers and UseTimeSync attribute values:

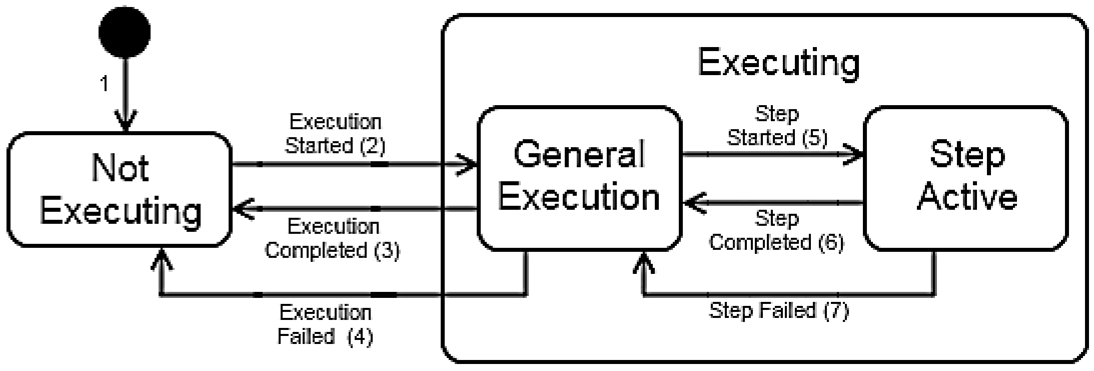
1. When the ICxTSObjectInfoCB::UseTimeSyncChanged callback is received from CIM148, enable or disable (depending on newVal parameter value) the NTP client software.
2. When the ICxTSObjectInfoCB::TimeServersChanged callback is received from CIM148, update the NTP client setting for the time servers.

9.4 Shutdown

Release CCxTSObject object instance.

10. CIM157

E157 Module Process Tracking standard provides a simple state machine to track the process of modules.



- Not Executing is the state the module is in when it is not executing a recipe
- General Execution is the state of the module when it is executing a recipe. For a recipe with no defined steps, it will remain in this state until recipe execution completes or fails. For a recipe with steps, the module will be in this state before the first step starts, between steps and after the last step completes or fails.
- Step Active is the state of the module when it is executing a recipe step.

10.1 EPJ File

The E157 epj file should be copied into the EPJ file for the equipment. Once that is done, there are adjustments that must be made to match the equipment. There are a set of status variables required for each Module Process on the equipment including: ModuleId_i, ModuleState_i, ModulePreviousState_i. Also, there are the following state change events: ExecutionStarted_i, ExecutionCompleted_i, ExecutionFailed_i, StepStarted_i, StepCompleted_i and StepFailed_i. There needs to be one set of these for each MPT module on the equipment. The 'i' is the index number suffix that corresponds to the subscriptIndex parameter used when the module is added during initialization.

10.2 Initialization

Follow the following procedure to initialize CIM157:

1. Create an instance of CCxE157Object object.
2. Call ICxE157Object Initialize method.
3. Optional: Implement the ICxMPTCallback interface and register an instance of it using ICxE157Object::RegisterCallback method to receive notifications about module state changes.
4. For each module process to track, call AddModule().This method takes moduleId and subscriptIndex parameters, both of which must be unique. The moduleId is used in all subsequent references to the module to manage state changes. The subscriptIndex is used in data values in the callback for callees to be able to identify which module is changed. New modules go through state change 1 and are put in the mptNotExecuting state.

10.3 Programming

A module's state changes are indicated in the E157 model with calls to the appropriate MPT prefixed methods. Each successful state change is accompanied by a callback to one of the methods on ICxMPTCallback and an event triggered in the client application.

10.3.1 State change 1

This transition occurs when a module is first created. The module is placed in the mptNotExecuting state. It only happens on startup and there is no explicit method of causing it.

10.3.2 Execution started (2)

To perform state change 2, call MPTExecutionStarted or MPTExecutionStarted2, depending on whether there is a single substrate or multiple substrates. The module must be in the mptNotExecuting state or an error is generated and the state is not changed. On success, the new state will be mptGeneralExecution. For a recipe with no defined recipes steps, make this transition when recipe execution begins.

10.3.3 Execution completed (3)

To perform state change 3, call MPTExecutionCompleted with the successful flag set to VARIANT_TRUE. The module must be in the mptGeneralExecution state or an error is generated and the state is not changed. On success, the new state will be mptNotExecuting. For a recipe with no defined recipes steps, make this transition when recipe execution completes.

10.3.4 Execution failed (4)

To perform state change 4, call MPTExecutionCompleted with the successful flag set to VARIANT_FALSE. The module must be in the mptGeneralExecution state or an error is generated and the state is not changed. On success, the new state will be mptNotExecuting. For a recipe with no defined recipes steps, make this transition when recipe execution completes with a failure.

10.3.5 Step started (5)

To perform state change 5, call MPTStepStarted. The module must be in the mptGeneralExecution state or an error is generated and the state is not changed. On success, the new state will be mptStepActive.

10.3.6 Step completed (6)

To perform state change 6, call MPTStepCompleted with the successful flag set to VARIANT_TRUE. The module must be in the mptStepActive state or an error is generated and the state is not changed. On success, the new state will be mptGeneralExecution.

10.3.7 Step failed (7)

To perform state change 7, call MPTStepCompleted with the successful flag set to VARIANT_FALSE. The module must be in the mptStepActive state or an error is generated and the state is not changed. On success, the new state will be mptGeneralExecution.

10.4 Callbacks

There is a single callback interface with a callback method for each state change. ExecutionStarted() is called when state change 2 occurs for a single substrate. ExecutionStarted2() is called when state change 2 occurs for multiple substrates. ExecutionCompleted() is called when state change 3 or 4 occurs. Differentiation between the two states is done with the successful flag. It is TRUE for state change 3 and FALSE for state change 4. StepStarted() is called when state change 5 occurs. StepCompleted() is called when state change 6 or 7 occurs. Differentiation between the two states is done with the successful flag. It is TRUE for state change 6 and FALSE for state change 7.

10.5 Shutdown

Call the Shutdown() method to allow all pending callback notifications to finish and remove any existing callbacks. Finally, release the CCxE157Object object instance.

11. CIM300Server

The CIM300Server object was designed for using CIM300 with 64-bit applications, but it may be used with 32-bit applications as well. Testing has shown a possible performance increase from this architecture. CIM300Server is a CIMConnect communications object that creates the CIM300 packages inside CIMConnect as described below.

11.1 Software Architecture Overview

The solution assumes the following software architecture which identifies the major modules in the proposed solution. Each of the modules is described below.

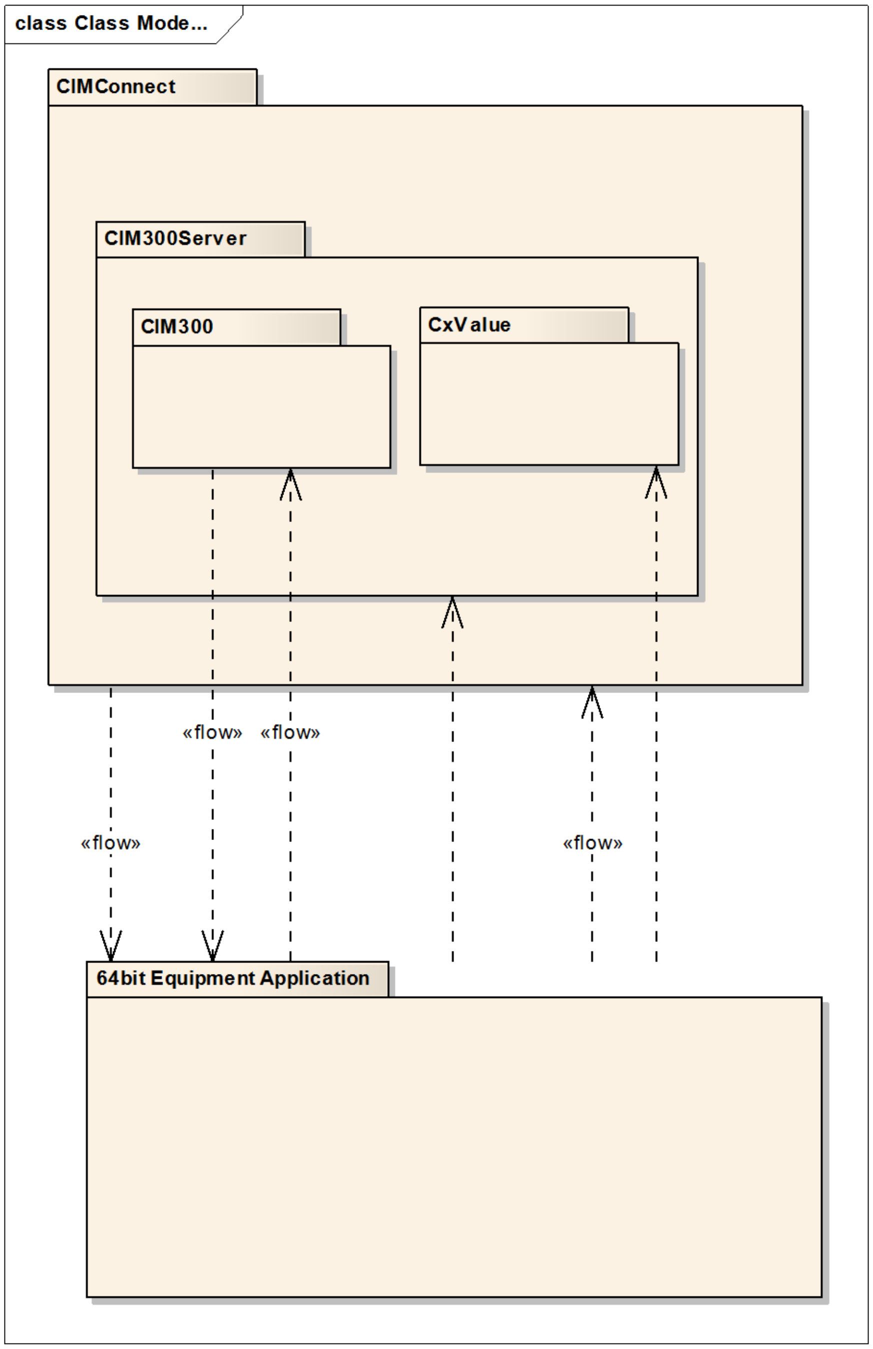


Figure 11-1 High Level Software Architecture Overview

11.1.1 64-bit Equipment Application

The 64-bit Equipment Application is the customer equipment control software. The following items are high-level requirements for the 64-bit Client Application.

11.1.2 CIMConnect

CIMConnect's EMService.exe process runs normally as a 32-bit process. The only change required beyond a normal CIMConnect configuration is the use of a new "host" connection and custom dll (CIM300Server) to house CIM300 and the Value object. CIMConnect's EPJ file needs to be modified to include a setting to load the CIM300 Application DLL as a Task DLL for a new connection.

11.1.3 CIM300Server

The CIM300Server is a custom CIMConnect communications object that loads CIM300 into EMService.exe. It is provided with CIM300 in SR13 or later. In this architecture CIM300Server acts as a CIM300 host and Value object server inside CIMConnect to get the CIM300 and Value 32-bit objects out-of-process so they may be used in a 64-bit application. CIM300Server is a COM object interface to allow the application to retrieve the CIM300 package interfaces.

11.1.4 CIM300

In this architecture, the CIM300 modules will be instantiated and live inside EMService in the CIM300Server. They will be accessed by the 64-bit Equipment Application through out-of-process COM calls.

11.1.5 CxValue

The CIM300Server provides an API to create Value objects. This API should be used by the 64-bit Equipment Application when calling APIs on CIM300 or CIMConnect that require a Value object. Do not use Value objects created with this API when interacting with the Application DCIM inside CIMPortal. This would cause unnecessary performance decreases as these Value objects are out-of-process to CIMPortal.

11.2 64-bit Equipment Application Interaction

11.2.1 CIMConnect Initialization

The only change to CIMConnect that is required is to edit the EPJ file to include a new host connection for the CIM300Server as shown below. The commifcfg setting for the CIM300Server is simply the number of the CIMConnect connection that should be used for SECS messaging. This is the connection the factory host is using and is passed into the Init method of the CIM300 abstraction layer. No events or parameters sections should be added for this connection.

```
[CONNECTION2]
#CIM300Server connection for 64bit application
name=CIM300Server
encoder=
decoder=
commif=CIM300Server.CIM300ServerCommIF
#If useregforcommifcfg is NOT zero then the commifcfg string will be read from the
registry and not the epj file.
useregforcommifcfg=0
#If forceupdatereg is NOT zero then the registry entry for this connection is
updated with the commifcfg string.
forceupdatereg=0
commifcfg=1
taskdll=
```

11.2.2 CIM300 Initialization

All that must change in CIM300 initialization is the code that creates the CIM300 package objects. Now the 64-bit Equipment Application will get the package interfaces from CIM300Server using one of the GetXX() API. The 64-bit Equipment Application may then use the interface just as if it had created the object. In the C# code below, you can see where GetCommIF on CIMConnect is used to get the ICxCommunications interface to the CIM300Server. Then you can see the call to GetAbstractionLayer() to get the CIM300 abstraction layer interface created by CIM300Server. This replaces what would normally be a “=new CCABSLAYERLib.CCAlObjectClass();”.

```
 //get the CIM300Server
 object o = svc.GetCommIF(0, cim300ServerConnectionId);
 cim300Server = (CIM300ServerLib.CIM300ServerCommIF)o;

 //Recreate the CIM300 Packages, this will recreate the cim300 package so they are
starting clean. Do not call this if you want to persist your CIM300 state across
application restarts.
 cim300Server.Recreate();

 //get the abstraction layer
 ICxAbsLayer pAbsLayer = (ICxAbsLayer)cim300Server.GetAbstractionLayer();
              pAbsLayer.Init

 //get the e39 package
 e39 = (CxE39Obj)cim300Server.GetE39();
 e39.Init2("top:top", pAbsLayer);
```

11.2.3 CIM300 Usage

One change in CIM300 usage is if an API has an ICxValueDisp* argument. If the API takes a value object, you will first need to get the Value object from the CIM300Server using the GetNewValueObject() method. You may then fill in the value object as needed and pass it into the CIM300 API.

Another change is that all CIM300 callbacks into your application will now occur on RPC threads because CIM300 is out-of-process to your application. This differs from a traditional, in-process implementation where some of the callbacks would be on application threads and some would be on RPC threads. In the traditional case, the RPC threads would be from CIMConnect messaging callbacks for GEM Host service requests. In this solution, all of the callbacks will be on RPC threads.

11.2.4 CIMConnect Usage

The only change in CIMConnect usage is if an API has an ICxValueDisp* argument. If the API takes a value object, you will first need to get the Value object from the CIM300Server using the GetNewValueObject() method. You may then fill in the value object as needed and pass it into the CIMConnect API. Interestingly, in this architecture using the xxxByteBuffer API of CIMConnect may be slower than the API using ICxValueDisp as the value object returned from GetNewValueObject are already in CIMConnect.

12. Appendix

The project file (.epj) describes the connection specifications, status variables, data variables, alarms, events etc. All variables, collection events and alarms per module id are captured in the subsequent sections of this document.

12.1 SEMI Standard ID (E40)/Cimetrix Module ID (CIM40)

12.1.1 Data Variables

Data Variable Name	Private	Persistent	User/ Product Handled	Developer Description
PRJobID	FALSE	FALSE	Product	The unique identifier for a process job. It can be accessed as the ObjID attribute of the process job. The host can provide this identifier to the processing resource. In this case, the host must guarantee uniqueness of job identifiers for all the process jobs in the PROCESS JOB STATE in the equipment. The processing agent assigns the unique identifier that is used in all subsequent process job communications.
PRJobStateAscii	FALSE	FALSE	Product	A text representation of the Process Job's state. This is not applicable with CE 8955 and 8987. For SR10 and later versions, PRJobStateAscii keeps its previous value for event reports that are associated with object extinction (to be compliant with the new E40 requirements).
PRJobState	FALSE	FALSE	Product	An enumeration of the unique sub-state of the job according to the process job state model. For SR10 and later versions, PRJobState keeps its previous value for event reports that are associated with object extinction (to be compliant with the new E40 requirements). PRSTATE enumerated as follows: 0 – QUEUED/POOLED 1 – SETTING UP 2 – WAITING FOR START 3 – PROCESSING 4 – PROCESS COMPLETE 5 – (Reserved) 6 – PAUSING 7 – PAUSED 8 – STOPPING 9 – ABORTING 10 – STOPPED 11 – ABORTED
PRJobPrevStateAscii	FALSE	FALSE	Product	The Process Job's previous state in text. This is not applicable with CE 8950 and 8970.
PRJobPrevState	FALSE	FALSE	Product	An enumeration of the Process Job's previous state.
PRJobRecID	FALSE	FALSE	Product	The Process Jobs' RecID attribute. Indentifier for the recipe applied.
PRJobPauseEvent	FALSE	FALSE	Product	The Process Job's PauseEvent attribute – It is a list of event identifiers that cause the equipment to automatically transition to the PAUSING/PAUSED states when one of the listed events is triggered.
PRJobPRMtlNameList	FALSE	FALSE	Product	The Process Job's PRMtlNameList attribute, where n is the number of carriers and j is the number of slots.

Data Variable Name	Private	Persistent	User/ Product Handled	Developer Description
PRJobPRMtlType	FALSE	FALSE	Product	Process Job's PRMtlType attribute. Identifies the type of material being processed. MaterialTypeEnum enumerated as follows: 1 – pjWAFER 2 – pjCASSETTE 3 – pjDIE_CHIP 4 – pjBOAT 5 – pjINGOT 6 – pjLEADFRAME 7 – pjLOT 8 – pjMAGAZINE 9 – pjPACKAGE 10 – pjPLATE 11 – pjTUBE 12 – pjWATERFRAME 13 – pjCARRIER 14 - pjSUBSTRATE
PRJobPRProcessStart	FALSE	FALSE	Product	Process Job's PRProcessStart attribute. Indicates that the processing resource start processing immediately when ready. TRUE – Automatic start FALSE – Manual start
PRJobPRRecipeMethod	FALSE	FALSE	Product	Process Job's PRRecipeMethod attribute. Indication of recipe specification type. RecipeMethodEnum enumerated as follows: 1 – pjRECIPE_ONLY 2 – pjRECIPE_WITH_VARIABLE_TUNING
PRJobRecVariableList	FALSE	FALSE	Product	The Process Job's RecVariableList attribute, where n is the number of reccipe variables. Valid values for the recipe variables, RCPPARNM and RCPPARVAL, are documented with SECS-II message S16, F3 Process Job Create Request.

12.1.2 Status Variables

Status Variable Name	Private	Persistent	User/ Product Handled	Developer Description
PRMtrlOrder	FALSE	TRUE	Product	An enumeration that defines the order by which material in the process jobs material list is processed. This is set by PRSetMtrlOrder S16, F29. . MaterialOrderEnum enumerated as follows: 1 – pjARRIVAL – process jobs are initiated according to the material arrival status. 2 – pjOPTIMIZE – Process jobs are initiated according to the equipment's resources. 3 – pjLIST – Process jobs are initiated according to the control job's list.

12.1.3 Equipment Constants

Equipment Constant Name	Private	Persistent	User/ Product Handled	Developer Description
PRJobEventStyle	FALSE	TRUE	Product	Determines which event style is used when there is an event for each state in the PJM state model. 0 Use the new style 1 Use the old style 2 Use both styles (default) The older event style, which was in place when CIM40 was originally developed, included the following events for each process job state: PJQueued - Process Job transitioned to QUEUED state PJSetup - Process Job transitioned to SETTING UP state PJWaitingForStart - Process Job transitioned to WAITINGFORSTART state PJProcessing - Process Job transitioned to PROCESSING state PJProcessingComplete - Process Job processing completed PJProcessJobComplete - Process Job completed PJPausing - Process Job transitioned to PAUSING state PJPaused - Process Job transitioned to PAUSED state PJStopping - Process Job transitioned to STOPPING state PJAborting - Process Job transitioned to ABORTING state These set of events do not specify the unique transition taken to reach a state. The CCS compliance test for PRJobEvents requires collection events for each state transition, so the following set of events were added to pass this test: PJ_tsm1 - Process Job No State to QUEUED/POOLED state PJ_tsm2 - Process Job QUEUED/POOLED state to SETTING UP state PJ_tsm3 - Process Job SETTING UP state to WAITING FOR START state PJ_tsm4 - Process Job SETTING UP state to PROCESSING state PJ_tsm5 - Process Job WAITING FOR START state to PROCESSING state PJ_tsm6 - Process Job PROCESSING state to PROCESS COMPLETE state PJ_tsm7 - Process Job PROCESS COMPLETE state to No State PJ_tsm8 - Process Job EXECUTING state to PAUSING state PJ_tsm9 - Process Job PAUSING state to PAUSED state PJ_tsm10 - Process Job PAUSE state to EXECUTING state PJ_tsm11 - Process Job EXECUTING state to STOPPING state PJ_tsm12 - Process Job PAUSE state to STOPPING state

Equipment Constant Name	Private	Persistent	User/ Product Handled	Developer Description
				PJ_tsm13 - Process Job EXECUTING state to ABORTING state PJ_tsm14 - Process Job STOPPING state to ABORTING state PJ_tsm15 - Process Job PAUSE state to ABORTING state PJ_tsm16 - Process Job ABORTING state to No State PJ_tsm17 - Process Job STOPPING state to No State PJ_tsm18 - Process Job QUEUED/POOLED state to No State With these additional events, the CCS test for PRJobEvents passes. The PRJobEventStyle variable is used to determine which set of events are to be used. There is different CCS test for PRJobAlerts that fails when the newer event style is chosen. The older event style is more appropriate for this test. If the older event style is chosen, PRJobEvents are not fully supported. If the newer event style is chosen, PRJobAlerts are not fully supported. Setting the value to 2 sends out both new AND old style events, and this is the most appropriate setting for passing the CCS compliance tests.
PRJobWaitForAllMaterial	FALSE	TRUE	Product	Determines if the Process Job waits for all material to arrive. 0 Wait for all material 1 Wait for any material
PRJobCheckSlotMapStatus Material	FALSE	TRUE	Product	Determines if the Process Job waits for the Slot Map Status of the Carriers to be verified. 0 Does not check 1 Waits for verification
SuppressPRJobAlert	FALSE	TRUE	Product	Provides ability to suppress the sending of PR Job Alerts. 1=No PRJobAlert messages will be sent 0=PRJobAlert messages will be sent. PR Job Alerts are notifications of process job milestones. Process job milestones are events which are important to the control and tracking of the process job. E40 defines the following PR Job Milestones: PR Job Setup PR Job Processing PR Job Processing Complete PR Job Complete PR Job Waiting for Start - used with the manual start option.
PRJobObjType	TRUE	TRUE	Product	E39 process job object OBJTYPE attribute value. “PROCESSJOB”
ExtendRecipeTuningState	TRUE	TRUE	Product	E40 doesn't specify the cases in which recipe parameter value changes sent from the Host in the PRJobSetRecipeVariable request (S16F23) should be rejected.

Equipment Constant Name	Private	Persistent	User/ Product Handled	Developer Description
				CIM300 is flexible in allowing the host to perform this job tuning operation and allows the equipment application to control whether parameter changes are accepted if it is implementing the PRSetRecipeVariable callback. This configuration setting can be used to allow CIM300 to either permit the host service request when the ProcessJob is in an Executing sub-state (and forward the request to the application callback) or automatically reject the request. 0 = Reject request (do not extend allowed states) 1 = Accept request/Call callback (extend allowed states to include Executing sub-states) default
E40ComplianceLevel	TRUE	FALSE	Product	Determines what level of compliance that CIM40 runs at. The default is always the latest version that is supported by Cimetrix. If you set E40ComplianceLevel to at least 200507, E94ComplianceLevel to at least 200603, and E40ProcessJobPersistence to at least 1, your jobs won't disappear when calling CCxE40PJM::PRJobComplete and CCxE94CJM::ProcessJobComplete. Instead, the jobs are deleted when the material is removed. Format: YYYYMM
E40ProcessJobPersistence	TRUE	TRUE	Product	Determines the PJ persistence. If you set E40ComplianceLevel to at least 200507, E94ComplianceLevel to at least 200603, and E40ProcessJobPersistence to at least 1, your jobs won't disappear when calling CCxE40PJM::PRJobComplete and CCxE94CJM::ProcessJobComplete. Instead, the jobs are deleted when the material is removed. 0001 PostActive(default) 0002 Super 0004 Sub 0010 Combined 0020 Data Ordered 0100 ExtinctOnActive 0200 ExtinctOnPostActive
PRJobMultiCreateContinueOnError	TRUE	TRUE	Product	Determines if valid jobs should be processed if an error occurs after requesting PRJobMultiCreate. 0 Stop processing 1 Continue processing (default) If 1, CIM300 continues processing the rest of the list of process jobs when a problem is detected in processing one of the process jobs in a PRJobMultiCreate (S16F15). Also, CIM300 sends a list of ERRCODE and ERRTEXT pairs in the S16F16 response corresponding to each process job

Equipment Constant Name	Private	Persistent	User/ Product Handled	Developer Description
				that had an error. If 0 (backward compatible with CIM300 versions before SR8), CIM300 stops processing the list when it detects the first error. Only one ERRCODE and ERRTEXT pair is sent in the reply.
PRJobCreateReturnOnError	TRUE	TRUE	Product	Determines if the Process Job returns the generated name on creation (even if there is an error). 0 – Do not return name on error 1 – Return name even on error(default)
AsuncPRStateChange	TRUE	TRUE	Product	Asynchronously call PRStateChange callback. 0=disabled 1=enabled

12.1.4 Collection Events

Event Name	Private	User/ Product Handled	Developer Description
PJ_tsm1	FALSE	Product	Process Job No State to QUEUED/POOLED state – Processing resource receives a Process Job create request. The job is placed in the job queue/pool. Process Job creation is acknowledged.
PJ_tsm2	FALSE	Product	Process Job QUEUED/POOLED state to SETTING UP state – The job is removed from the queue/pool. PR Job Setup event is triggered. All required resource preconditioning is performed. When job material arrives all material preparation is performed.
PJ_tsm3	FALSE	Product	Process Job SETTING UP state to WAITING FOR START state – Job material is present AND the process resource is ready to start the process job AND PRProcessStart attribute is not set. PR Job is Waiting for Start event triggered.
PJ_tsm4	FALSE	Product	Process Job SETTING UP state to PROCESSING state – Material is present and ready for processing. PRProcessStart attribute is set. PR Job Processing event is triggered. Material is processed.
PJ_tsm5	FALSE	Product	Process Job WAITING FOR START state to PROCESSING state – Job Start directive received. PR Job Processing event is triggered. Material is processed.
PJ_tsm6	FALSE	Product	Process Job PROCESSING state to PROCESS COMPLETE state – Material processing completes. PR Job Processing Complete event is triggered. The processing resource performs all required resource post-conditioning. Await material departure.
PJ_tsm7	FALSE	Product	Process Job transitions from any POST-ACTIVE state to No State – Job material departs from the equipment OR the process job becomes extinct because the process job is replaced by another process job that specifies the same material when no control job is used. The Process Job Complete event is triggered and the job is deleted.
PJ_tsm8	FALSE	Product	Process Job EXECUTING state to PAUSING state – Processing resource

Event Name	**Private**	**User/ Product Handled**	**Developer Description**
			initiates a process pause action. The process resource pauses at the first convenient time.
PJ_tsm9	FALSE	Product	Process Job PAUSING state to PAUSED state – The processing resource successfully pauses the job
PJ_tsm10	FALSE	Product	Process Job PAUSE state to EXECUTING state – The processing successfully resumes paused activity.
PJ_tsm11	FALSE	Product	Process Job EXECUTING state to STOPPING state – The processing resource initiates a process stop action. The processing resource stops the current activity at the first convenient time.
PJ_tsm12	FALSE	Product	Process Job PAUSE state to STOPPING state -- Processing resource initiates a process stop action. The processing resource stops the current activity at the first convenient time.
PJ_tsm13	FALSE	Product	Process Job EXECUTING state to ABORTING state -- Processing resource initiates a process abort action. The processing resource terminates the current activity immediately.
PJ_tsm14	FALSE	Product	Process Job STOPPING state to ABORTING state -- Processing resource initiates a process abort action. The processing resource terminates the current activity immediately.
PJ_tsm15	FALSE	Product	Process Job PAUSE state to ABORTING state -- Processing resource initiates a process abort action. The processing resource terminates the current activity immediately.
PJ_tsm16	FALSE	Product	Process Job ABORTING state to ABORTED State – The abort procedure is completed.
PJ_tsm17	FALSE	Product	Process Job STOPPING state to STOPPED state – The stop procedure is completed and all material is in a safe condition.
PJ_tsm18	FALSE	Product	Process Job QUEUED/POOLED state to No State – “CANCEL”, “ABORT” or “STOP” command received. Process job is removed from queue/pool. PR Job Complete event is triggered. Process Job is deleted.
PJQueued	FALSE	Product	Process Job transitioned to QUEUED state
PJSetup	FALSE	Product	Process Job transitioned to SETTING UP state.
PJWaitingForStart	FALSE	Product	Process Job transitioned to WAITINGFORSTART state.
PJProcessing	FALSE	Product	Process Job transitioned to PROCESSING state.
PJProcessingComplete	FALSE	Product	Process Job processing completed.
PJProcessJobComplete	FALSE	Product	Process Job completed.
PJPausing	FALSE	Product	Process Job transitioned to PAUSING state.
PJPaused	FALSE	Product	Process Job transitioned to PAUSED state.
PJStopping	FALSE	Product	Process Job transitioned to STOPPING state.
PJStopped	FALSE	Product	Process Job transitioned to ABORTING state.
PJAborting	FALSE	Product	Process Job transitioned to Post-Active STOPPED state.
PJAborted	FALSE	Product	Process Job transitioned to Post-Active ABORTED state.

12.2 SEMI Standard ID (E87)/Cimetrix Module ID (CIM87)

12.2.1 Data Variables

Data Variable Name	Private	Persistent	User/ Product Handled	Developer Description
AccessMode	FALSE	FALSE	Product	An enumeration that defines the host view of the equipment access mode, as well as the host interactions with the equipment that is necessary to switch the access mode. 0 = MANUAL 1 = AUTO Each Load Port has its own Access Mode State Model. The access mode for a load port can be switched at any time by the host or the operator (except when the Load Port Reservation State Model for that Load Port is in the RESERVED state or during carrier transfer). Used by S3F27 to specify the desired Port Access Mode. Range: U1 0-U1 255 Default: U1
AvailPartitionCapacity	FALSE	FALSE	Product	The current available buffer capacity for a logical partition that is inside the internal buffer equipment (applicable only to internal buffer production equipment). The partition capacity is the number of carriers in the partition. A change in the AvailablePartitionCapacity causes CIM300 to automatically trigger the BufferCapacityChanged Event.
BufferPartitionInfo	FALSE	FALSE	Product	The related information for a logical buffer partition that is contained in a structure that has the following items: PartitionID, PartitionType, AvailPartitionCapacity, PartitionCapacity, and UnallocatedPartitionCapacity (only applicable to internal buffer production equipment). Range: L-L Snytax: L, 5 1. <PartitionID> 2. <PartitionType> 3. <AvailPartitionCapacity> 4. <PartitionCapacity> 5. <UnallocatedPartitionCapacity>
CapacityDV	FALSE	FALSE	Product	The carrier capacity.
CarrierAccessingStatus DV	FALSE	FALSE	Product	An enumeration of the current accessing state of the carrier by the equipment or the current substate of the CarrierAccessingStatus state model. The value is used by the host to know whether the carrier that is owned by the equipment can be moved out. If the carrier is within the internal

Data Variable Name	Private	Persistent	User/ Product Handled	Developer Description
				buffer equipment, the status can be used by the host to issue CarrierOut service. In all cases, the state must be set to either CARRIER COMPLETE or CARRIER STOPPED before the carrier may be undocked. 0 = NOT_ACCESSED 1 = IN_ACCESS 2 = CARRIER_COMPLETE 3 = CARRIER_STOPPED. Range: U1 0-U1 255 Default: U1
LocationID	FALSE	FALSE	Product	The identifier of the current location. Carrier locations are any location at or in the production equipment where a carrier can rest. It conforms to restrictions of ObjID as specified in SEMI E39.1, Section 6. Format: 1–80 characters
CarrierID	FALSE	FALSE	Product	A readable and unique identifier for the carrier. If there are no currently existing objects with the carrierID that has been read, a carrier object is instantiated when the equipment successfully reads the carrierID. If the CarrierID is provided before the equipment reads the CarrierID, the CarrierID that becomes associated with the load port can be used later for equipment based carrier verification. CarrierID conforms to restrictions of ObjID, as defined in SEMI E39.1, Section 6. Format: 1-80 characters
PortID	FALSE	FALSE	Product	The ID number of a load port. The PortID number should be the same as the load port number.
CarrierIDStatusDV	FALSE	FALSE	Product	An enumeration of the current state of the carrier ID verification status. 0 = ID_NOT_READ, 1 = ID_WAITING_FOR_HOST, 2 = ID_VERIFICATION_OK, 3 = ID_VERIFICATION_FAILED
ContentMapDV	FALSE	FALSE	Product	Ordered list of lot and substrate identifiers that correspond to slot 1,2,3,…n. Each structure consists of a LotID and SubstrateID. When no Lot ID is provided by the host, the LotID value should be null.

Data Variable Name	**Private**	**Persistent**	**User/ Product Handled**	**Developer Description**
				When a slot has no substrate or the host does not know substrate identifier, the LotID and SubstrateID value should be null. SubstID conform to the restrictions of ObjID as specified in SEMI E39.1, Section 6. The ContentMap attribute is an empty list (a list of zero) when the instantiation is done by CarrierID read. Format: L, n n=Capacity 1. L,2 1. 20 (A) LotID 2. 20 (A) SubstID … n. L,2 1. 20 (A) LotID 2. 20 (A) SubstID
LoadPortReservationState	FALSE	FALSE	Product	An enumeration of the Load Port reservation state. 0 = LP_NOT_RESERVED 1 = LP_RESERVED
PartitionCapacity	FALSE	FALSE	Product	The total PartitionCapacity for a logical internal buffer partition. It is applicable only to internal buffer production equipment.
PartitionID	FALSE	FALSE	Product	The ID of a logical internal buffer partition, which is used to identify separate material types in an internal buffer. It conforms to restrictions of ObjID as specified in SEMI E39.1, Section 6. Format: 1-80
PartitionType	FALSE	FALSE	Product	The type of a logical partition within an internal buffer. It is applicable only to internal buffer production equipment. Some examples of logical buffer types are Product, Dummy, Substrate, and Seed. Format: 1-64
PortAssociationState	FALSE	FALSE	Product	An enumeration of the association state of a load port. 0 = NOT_ASSOCIATED 1 = ASSOCIATED
PortGroup	FALSE	FALSE	Product	Port Group Format: L,3 1. <PORTGRPNAME> 2. <PORTACCESS> 3. L,n 1. <PTN$_1$>

Data Variable Name	**Private**	**Persistent**	**User/ Product Handled**	**Developer Description**
				. . n. <PTN$_n$>
PortGroupID	FALSE	FALSE	Product	Port Group ID
PortListDV	FALSE	FALSE	Product	List of ports. Format: L, n PortID$_1$ . . PortID$_n$
PortStateInfo	FALSE	FALSE	Product	The PortAssociationState combined with the PortTransferState. Format: L,2 1. <PortAssociationState> 2. <PortTransferState>
PortTransferState	FALSE	FALSE	Product	An enumeration of the current transfer state of a load port. Super states are not included; it only includes the sub states. 0 = LP_OUT_OF_SERVICE 1 = LP_TRANSFER_BLOCKED 2 = LP_READY_TO_LOAD 3 = LP_READY_TO_UNLOAD 4 = LP_IN_SERVICE 5 = LP_TRANSFER_READY
Reason	FALSE	FALSE	Product	An enumeration for the reason for transition 14, SLOT MAP NOT READ to WAITING FOR HOST. This information aids the host in deciding the appropriate action. 0 = rVERIFICATION_NEEDED 1 = rVERIFICATION_BY_EQ_UNSUCCESSFUL 2 = rREAD_FAIL 3 = rIMPROPER_WAFER_POSITION Range: 0-255 Default: 0
SlotMapDV	FALSE	FALSE	Product	The slot map of a carrier. Ordered list of n, where n is equal to the value of the Capacity attribute of the carrier. Each value in the list is from the enumeration defined for the attribute SlotMap. Format: L, n 1. Slotmap status <U1 > 2. Slotmap status <U1 > 3. Slotmap status <U1 > 4. Slotmap status <U1 > 5. …….

Data Variable Name	**Private**	**Persistent**	**User/ Product Handled**	**Developer Description**
				6. ……. n. ……. n = capacity
SlotMapStatusDV	FALSE	FALSE	Product	The carrier slot map verification state. 0 - slotmap is not read. 1 - slotmap is waiting for the host. 2 - slotmap is verified. 3 - slotmap verification has failed.
SubstrateCountDV	FALSE	FALSE	Product	The carrier substrate count.
UnAllocatedPartitionCapacity	FALSE	FALSE	Product	In CIM87, CareateCarrier, ProceedWithCarrier, CancelCarrier, Bind, CarrierAtPort, ReserveAtPort, CancelCarrierNotification APIs update this dv.
UsageDV	FALSE	FALSE	Product	The carrier substrate usage.
UsageDefault	FALSE	FALSE	Product	CxE87Carrier::Init function will set the ASCII value of this DV to the UsageDV only when it exists.

12.2.2 Status Variables

Status Variable Name	**Private**	**Persistent**	**User/ Product Handled**	**Developer Description**
AccessMode_[i]	FALSE	TRUE	Product	An enumeration that defines the access mode for the ith load port. 0 = MANUAL 1 = AUTO
AvailPartitionCapacity_[i]	FALSE	FALSE	Product	The current available buffer capacity for the ith logical partition that is inside the internal buffer equipment (applicable only to internal buffer production equipment). The partition capacity is the number of carriers in the partition.
BufferCapacityList	FALSE	FALSE	Product	The list of all current PartitionType, AvailPartitionCapacity, and PartitionCapacity for all logical buffer partitions (applicable only to internal buffer production equipment). Range: L-L Format: L,n 1. <BufferPartitionInfo1> . . n.<BufferPartitionInfon>
BufferPartitionInfo_[i]	FALSE	FALSE	Product	The related information for the ith buffer partition that is contained in a structure that has the following items: PartitionIDi, PartitionTypei, AvailPartitionCapacityi, PartitionCapacityi, and UnallocatedPartitionCapacity (only

Status Variable Name	**Private**	**Persistent**	**User/ Product Handled**	**Developer Description**
				applicable to internal buffer production equipment). Range: L-L Format: L, 5 1. <PartitionID> 2. <PartitionType> 3. <AvailPartitionCapacity> 4. <PartitionCapacity> 5. <UnallocatedPartitionCapacity>
CarrierID_[i]	FALSE	FALSE	User	A readable and unique identifier for the carrier at the ith locationID. CarrierID conforms to restrictions of ObjID, as defined in SEMI E39.1, Section 6. Format: 1-80 characters
CarrierLocationMatrix	FALSE	FALSE	Product	A list of all the carriers at or in the equipment (both internal to the equipment and on equipment load ports). It is a list of n pairs of items. If there is no carrier at the locationIDi, the CarrierIDi is NULL. If a carrier is at LocationIDi but the CarrierIDi is not known, the value of CarrierIDi is "UNKNOWN." Format: L,n 1. L,2 1. <LocationID1> 2. <CarrierID1> . . n. L,2 1. <LocationIDn> 2. <CarrierIDn>
LoadPortReservationState_[i]	FALSE	FALSE	Product	An enumeration of the Load Port reservation state. 0 = Not Reserved 1 = Reserved
LoadPortReservationState List	FALSE	FALSE	Product	Format: L,n 1.<LoadPortReservationState1> . . n.<LoadPortReservationStateN>
LocationID_[i]	FALSE	FALSE	Product	The LocationID of the ith carrier location. Carrier locations are any location at or in the production equipment where a carrier can rest.

Status Variable Name	**Private**	**Persistent**	**User/ Product Handled**	**Developer Description**
				It conforms to restrictions of ObjID as specified in SEMI E39.1, Section 6. Format: 1-80 characters
PartitionCapacity_[i]	FALSE	FALSE	Product	The PartitionCapacity for the ith PartitionID of the internal buffer. It is applicable only to internal buffer production equipment.
PartitionID_[i]	FALSE	FALSE	Product	The ID of the ith logical partition of the internal buffer, which is used to identify separate material types in an internal buffer. It conforms to restrictions of ObjID as specified in SEMI E39.1, Section 6. Format: 1-80
PartitionType_[i]	FALSE	FALSE	Product	The type of the ith logical partition within an internal buffer. It is applicable only to internal buffer production equipment. Some examples of logical buffer types are Product, Dummy, Substrate, and Seed. Format: 1-64
PortAssociationState_[i]	FALSE	FALSE	Product	An enumeration of the association state of the ith load port. 0 = NOT_ASSOCIATED 1 = ASSOCIATED
PortID_[i]	FALSE	FALSE	User	ID of the ith load port where the carrier transfer is taking place. One PortID exists for each load port.
PortGroupList	FALSE	FALSE	Product	Port Group List
PortAssociationStateList	FALSE	FALSE	Product	A list of the current association states for all load ports. This can be used to resynchronize the host. Format: L,n 1. <PortAssociationState1> . . n. <PortAssociationStaten>
PortList	FALSE	FALSE	Product	A list of Port IDs. Format: PortID1 . . n PortIDn
PortStateInfo_[i]	FALSE	FALSE	Product	The PortAssociationState combined with the PortTransferState for the ith load port. It's a list of 2 items: * PortAssociationStatei * PortTransferStatei

Status Variable Name	**Private**	**Persistent**	**User/ Product Handled**	**Developer Description**
				Format: L,2 1. <PortAssociationStatei> 2. <PortTransferStatei>
PortStateInfoList	FALSE	FALSE	Product	A list of all of the port states for all load ports. Format: L,n 1. <PortStateInfo1> . n. <PortStateInfon>
PortTransferState_[i]	FALSE	TRUE	Product	An enumeration of the current transfer state of the ith load port. Super states are not included; it only includes the sub states. 0 = LP_OUT_OF_SERVICE 1 = LP_TRANSFER_BLOCKED 2 = LP_READY_TO_LOAD 3 = LP_READY_TO_UNLOAD 4 = LP_IN_SERVICE 5 = LP_TRANSFER_READY
PortTransferStateList	FALSE	FALSE	Product	The current Load Port Transfer State for all load ports. This can be used to resynchronize the host. Format: L,n 1. <PortTransferState1> . . n. <PortTransferStaten>
SubstrateIDSeparator	FALSE	FALSE	Product	This is used when generating substrate IDs from a slot map.
UnAllocatedPartitionCapacity_[i]	FALSE	FALSE	Product	In CIM87, CareateCarrier, ProceedWithCarrier, CancelCarrier, Bind, CarrierAtPort, ReserveAtPort, CancelCarrierNotification APIs update this dv.

12.2.3 Equipment Constants

Equipment Constant Name	**Private**	**Persistent**	**User/ Product Handled**	**Developer Description**
CMSMMismatchHandling	FALSE	TRUE	Product	It is possible that the host can send a ContentMap in the ProceedWithCarrier that conflicts with the SlotMap. This configuration option provides three alternative solutions for CIM300's automatic handling of this scenario and allows the user to select the preferred solution. 0 - Do not reject the ProceedWithCarrier service request from the host if the ContentMap and SlotMap do not match. CIM300 uses the content map to build the substrates (unless UseSlotMapOnly is set to TRUE).

Equipment Constant Name	**Private**	**Persistent**	**User/ Product Handled**	**Developer Description**
				1 - Reject the ProceedWithCarrier service request from the host if the ContentMap and SlotMap do not match. 2 - Do not reject the ProceedWithCarrier service request from the host if the ContentMap and SlotMap do not match. CIM300 automatically alters the ContentMap as follows: For each slot, assuming that the number of entries in the content map and slot map match: A) If the slot map is occupied correctly and the content map is empty, create content map entries and substrates (content map is altered). B) If the slot map is unoccupied and the content map is populated, remove the entries from the content map and do not create a substrate (content map is altered). C) If the slot map is incorrectly occupied and the content map is empty, create content map entries and substrates (content map is altered).
E87AsyncProceedWithCarrier	FALSE	TRUE	Product	This setting enables the S3F17 ProceedWithCarrier messages to process asynchronously in CIM300. When the message comes in, the data is captured, a new thread is started to perform ProceedWithCarrier, and control is returned back to CIMConnect--immediately allowing it to process other incoming and outgoing messages while the substrates are being created. For backwards compatibility, the default is 0 if they do not exist. If the setting is non-zero, it executes the appropriate function in a separate asynchronous thread. Setting this value to 1 is recommended only for applications that experience performance problems during the ProceedWithCarrier operation.
E87AsyncCarrierRecreate	FALSE	TRUE	Product	This setting enables the S3F17 CarrierReCreate messages to process asynchronously in CIM300. When the message comes in, the data is captured, a new thread is started to perform CarrierReCreate, and control is returned back to CIMConnect--immediately allowing it to process other incoming and outgoing messages while the substrates are being created. For backwards compatibility, the default is 0 if they do not exist. If the setting is non-zero, it executes the appropriate function in a separate asynchronous thread. Setting this value to 1 is recommended only for applications that experience performance problems during the

Equipment Constant Name	Private	Persistent	User/ Product Handled	Developer Description
				CarrierReCreate operation.
ProceedWithCarrierBypass Create	FALSE	TRUE	Product	This configuration option changes the behavior of CIM300 in a scenario where a carrier object exists before the carrier arrives at the equipment (CarrierNotification or Bind) and a CarrierID reader is not available. Here is an example of such a scenario: 1. Host initiates CarrierNotification with CarrierID = CARRIER1 2. Carrier Arrives, but the Carrier ID reader is not available, so the CIM300 CarrierAtPort() is called by equipment application with an empty string for CarrierID and VARIANT_FALSE for reader valid. The result from the call is that the carrier object is in the WAITING_FOR_HOST state. 3. The Host (or application) performs a ProceedWithCarrier with CarrierID = CARRIER1 The value of this configuration setting will determine the outcome: 0 - (backward compatible with CIM300 versions before SR8) ProceedWithCarrier fails in this case and CIM300 reports a duplicate CarrierID. 1 - ProceedWithCarrier succeeds for existing CARRIER1 object.
ExtraProceedWithCarrier	FALSE	TRUE	Product	This configuration setting is for a specific type of scenario where the host system mistakenly sends an extra ProceedWithCarrier request after the CarrierID has already verified (but before the CarrierSlotMapStatus is WAITING FOR HOST). The following is an example of such a scenario (Host Based Verification is assumed): 1. A carrier arrives at the load port, and CIM300 CarrierAtPort is called with carrierID = carrier1. 2. The host sends ProceedWithCarrier with carrierID = carrier1. 3. The carrier on the load port is moved into the equipment (internal buffer). 4. The load port is now empty, but the host mistakenly sends another ProceedWithCarrier with carrierID = carrier1 In this case - ProceedWithCarrier is not expected. - There is no carrier on the load port. - The carrier already exists. - The carrierID has already been verified.

Equipment Constant Name	Private	Persistent	User/ Product Handled	Developer Description
				This configuration option determines how CIM300 handles the request: 0 - (Backward compatible with versions of CIM300 before SR8.) CIM300 calls the application's ProceedWithCarrier callback and sends CAACK = 0 in S3F18 unless the callback returns E_FAIL. 1 - CIM300 rejects the request, and the S3F18 reply contains a CAACK = 3, ERRCODE = 17, and ERRTEXT = No carrier at load port.
E87ERRCODEFormat	FALSE	TRUE	Product	This configuration option allows the user to select different ERRCODE data formats in reply messages for E87. The types are U1 (=1) U2 (=2) U4 (=4) U8 (=0)
BypassReadID	FALSE	TRUE	Product	A variable that enables or disables automatic ID acceptance and bypasses verification of the carrier ID when the carrier ID reader is unavailable or not installed and the loadport is ASSOCIATED. When TRUE, the Carrier ID that was received in the Bind is used automatically. Otherwise, the carrier transitions to WAITING FOR HOST and waits for the host to send a ProceedWithCarrier. The ID is the ID that was included with the ProceedWithCarrier method. TRUE FALSE Default: FALSE
BypassReadID_[i]	FALSE	TRUE	Product	A variable that enables or disables automatic ID acceptance and bypasses verification of the carrier ID when the carrier ID reader is unavailable or not installed for loadport i. TRUE FALSE Default: FALSE
CallCallbackFirst	FALSE	TRUE	Product	A value that specifies whether the CancelCarrier callback is called before any state changes are made to the carrier. 0 Default behavior 1 Call before
CarrierHold	FALSE	TRUE	User	If Carrier Hold is set to Host Release, both fixed buffer and internal buffer equipment hold the carrier at the write position until the CarrierRelease service is received, regardless of when a CarrierOut request is sent. If CarrierHold is set to Equipment Release, the equipment releases the carrier based on the Carrier Object state model transition to CARRIER

Equipment Constant Name	Private	Persistent	User/ Product Handled	Developer Description
				COMPLETE or CARRIER STOPPED and the CarrierRelease request has no effect. The host is able to both read and write whenever the CarrierHold switch is set to Host Release and the carrier is at the respective read or write position. Once the host has completed all of its read and write operations for that carrier, the host sends the CarrierRelease request to the equipment. 0 Host Release 1 Equipment Release after Carrier Complete/Stopped Range: U1 0-U1 1
CarrierRecreateLPSMStyle	TRUE	TRUE	Product	The CarrierReCreate is supposed to produce 1 of 3 possible Carrier Object State Model (COSM) transition events based on E87-0705: 1- Transition Event 3 (WaitingForHost) if no PropertiesList is provided with the CarirerReCreate service 2- Transition Event 2 (IDNotRead) if PropertiesList is provided and a bypassIDReader is False 3- Transition Event 4 (Verification OK) if PropertiesList is provided and BypassIDReader is True. The CarrierRecreateLPSMStyle configuration option provides support for this requirement. 1= COSM-3 is produced if no parameter list is given, while COSM-2 is produced if there is. COSM-4 is also produced if there is an invalid ID reader and BypassReadID is also true. 0= Compatibility with CIM300 versions before SR8 P2. Only COSM-2, or 'ID Read Failed' events are produced. If this variable is not defined in the EPJ file, the default value is 1.
CarrierUnclamp	FALSE	TRUE	User	0=Carrier Complete/Stopped triggered 1=AMHS triggered
CEIDSelectorCSM4	FALSE	TRUE	Product	0=CSM transition 4 CE 1=CSM transition 8 CE (Intel's req.) is triggered when CIDR unavailable
CEIDSelectorCSM5	FALSE	TRUE	Product	0=CSM transition 5 CE 1=CSM transition 9 CE (Intel's req.) is triggered when CIDR is unavailable
CMSSlotMapDVHandling	TRUE	TRUE	Product	When VerifySlotMap is called with an empty slot map, meaning that the slot map read failed: 0 = set the SlotMapDV to an empty list value 1 = set the SlotMapDV to a list of UNDEFINED (0) slot map values. This is the default.
E87ErrID_CCX01	TRUE	TRUE	Product	This variable allows the user to customize the error code and text for a case where a CancelCarrier request (S3F17) is

Equipment Constant Name	Private	Persistent	User/ Product Handled	Developer Description
				received by CIM87, but the carrier ID is either invalid or does not exit. The ASCII value configured in the EPJ file is a combination of the error code value for the ERRCODE data item and the text value for the ERRTEXT data item that is used in the S3F18 reply (with a colon (:) as a delimeter). Recommended setting: 3:Unknown object instance - Unknown CarrierID Backwards Compatibility value: 50:Missing Carrier - Carrier does not exist, loadport not occupied
E87ErrID_CCX02	TRUE	TRUE	Product	This variable allows the user to customize the error code and text for a case where a CancelCarrier request (S3F17) is received by CIM87, but the callback implementation rejects the request. The ASCII value configured in the EPJ file is a combination of the error code value for the ERRCODE data item and the text value for the ERRTEXT data item that is used in the S3F18 reply (with a colon (:) as a delimeter). Recommended setting: 22:Application didn't process CancelCarrier
E87ErrID_PWC04	TRUE	TRUE	Product	This configuration setting can be used to change the values sent to the host when a ProceedWithCarrier (S3F17) is received from the host with an incorrect CarrierID. Because different users and host systems might prefer different error codes and/or error text, CIM300 includes this variable to allow the user to configure the values for the ERRCODE and ERRTEXT data items in the S3F18 message. The ASCII value configured in the EPJ file is a combination of the error code value for the ERRCODE data item and the text value for the ERRTEXT data item that is used in the S3F18 reply (with a colon (:) as a delimeter). Recommended setting: 3:Unknown object instance - Unknown CarrierID Backwards Compatibility value: 17:Command not valid for current state - Specified carrier not present at loadport
E87ErrID_PWC05	TRUE	TRUE	Product	This configuration setting can be used to change the values sent to the host when a ProceedWithCarrier (S3F17) is received and the callback implementation wants the application to reject it. Because different users and host systems might prefer different error codes and/or error text, CIM300 includes this variable to allow the user to configure the values for the ERRCODE and ERRTEXT data items in the S3F18 message. The ASCII value configured in the EPJ file is a combination of the error code value for the ERRCODE data item and the

Equipment Constant Name	**Private**	**Persistent**	**User/ Product Handled**	**Developer Description**
				text value for the ERRTEXT data item that is used in the S3F18 reply (with a colon (:) as a delimeter). Recommended setting: 22:Application didn't process ProceedWithCarrier
E87ErrID_ BIND02	TRUE	TRUE	Product	This configuration setting can be used to change the values sent to the host when a Bind (S3F17) is received and the callback implementation wants the application to reject it. Because different users and host systems might prefer different error codes and/or error text, CIM300 includes this variable to allow the user to configure the values for the ERRCODE and ERRTEXT data items in the S3F18 message. The ASCII value configured in the EPJ file is a combination of the error code value for the ERRCODE data item and the text value for the ERRTEXT data item that is used in the S3F18 reply (with a colon (:) as a delimeter). Recommended setting: 22:Application didn't process BIND
E87ErrID_CARNOT02	TRUE	TRUE	Product	This configuration setting can be used to change the values sent to the host when a CARRIERNOTIFICATION (S3F17) is received and the callback implementation wants the application to reject it. Because different users and host systems might prefer different error codes and/or error text, CIM300 includes this variable to allow the user to configure the values for the ERRCODE and ERRTEXT data items in the S3F18 message. The ASCII value configured in the EPJ file is a combination of the error code value for the ERRCODE data item and the text value for the ERRTEXT data item that is used in the S3F18 reply (with a colon (:) as a delimeter). Recommended setting: 22:Application didn't process CARRIERNOTIFICATION
E87SMVerifiedEventAfter SubstrateCreate	TRUE	TRUE	Product	Allows the user to determine when the SlotMapVerificationOK event is triggered: Before the substrates are created (0 - default), or after (1). In CIM87, VerifySlotMap() compares the slot map from the host to the slot map that is generated by the equipment. If the two strings match, the carrier changes to SlotMapVerificationOK and triggers an event. Then, it proceeds to create the substrates. Some host systems use this event as a trigger to create the Process and Control jobs. If the control job is set to Autostart, a conflict can arise between E87 (creating substrates) and any callback that needs to verify that the substrates exist. The result is delayed message traffic.
PWCCheckSlotMap	FALSE	TRUE	Product	Determines whether the slot map is checked for cross-slotted or double-slotted wafers and rejects the service if one is

Equipment Constant Name	**Private**	**Persistent**	**User/ Product Handled**	**Developer Description**
				found. For Bind and ProceedWithCarrier. 0 Don't check slot map 1 Check slot map
SendCIDVerifyAlarm	FALSE	TRUE	Product	Determines whether to send a CID Verify Alarm on CarrierAtPort when the ID does not match the Associated ID. 0 Do not send on CarrierAtPort (default) 1 Send on CarrierAtPort
TrimCarrierIDs	TRUE	TRUE	Product	Some CarrierID readers produce CarrierID strings with leading and/or trailing whitespace, and some host systems send SECS messages with CarrierIDs containing these whitespace characters. SEMI E39 does not allow Object IDs (including CarrierIDs) to contain these extra whitespace characters. This configuration setting allows CarrierIDs with leading and trailing whitespace by trimming the whitespace from the beginning and end of the CarrierIDs in CIM87, process jobs, and control jobs. 0 Reject CarrierIDs with whitespace (default) 1 Allow CarrierIDs with leading or trailing whitespace and trim those characters from the CarrierID value stored in CIM300 To retain compliance to the SEMI E39 and E87 standards, use setting 0. Use 1 only if it's required by the fab host system.
UseCarrierPlaced	FALSE	TRUE	Product	0 CarrierAtPort 1 CarrierPlaced
UseSavedTransferState	TRUE	TRUE	Product	Determines how to figure out the IN SERVICE substate for ChangeServiceStatus. 0 Use old behavior for determining IN SERVICE substate 1 Use saved load port transfer state when going to IN SERVICE
UseSlotmapOnly	FALSE	TRUE	Product	This configuration option determines whether or not CIM300 uses the ContentMap for the carrier object when creating the corresponding substrate objects. If TRUE (1), only the SlotMap information is used. Otherwise, CIM300 evaluates the ContentMap and uses that information for substrate creation. This affects the substrate IDs that are generated. The ContentMap can contain specific substrate IDs (for example, specified by the Host). However, if this setting is TRUE, CIM300 automatically generates the substrate IDs according to the SEMI E90 requirements (which state that the default substrate ID is represented in text as the carrierID, a period,

Equipment Constant Name	Private	Persistent	User/ Product Handled	Developer Description
				and the substrate slot location). For example, if CarrierID = “xyz” and substrate slot location = “5,” the substrate ID is = “xyz.05.” For this reason and if UseSlotmapOnly is TRUE, the input parameters to GetSubstrateFromSource must specify the substrate ID in the <carrier.slot> format. Default: 0
UseSlotmapWithContentMap	FALSE	TRUE	Product	If TRUE, use the slotmap when creating substrates from the contentmap (to check if slot is occupied correctly).
ReservedPortStatus	FALSE	TRUE	Product	ReserveAtPort API checks this equipment constant value and generate a custom error message when the port is not reserved and the value is set to ‘1’ and the port is not reserved. Default: 0
PWCCarrierIDType	TRUE	TRUE	Product	The product will reject PWC (S3F17) with an S9F7or S3F18 depending on the value of this equipment constant. ‘1’ means reject with an S9F7 message when the carried id type is mismatched. ‘0’ means reject with an S3F18 message when the carried id type is mismatched. Default: 0
SendAlarmsOnCancel	FALSE	TRUE	Product	When id verification or slot map verification failed the product will generate an alarm in CancelCarrier and CancelCarrierAtPort APIs depending on the EC value: 0 – send the alarm 1 – don’t send the alarm Default: 1
SendSlotMapVerifyAlarm	FALSE	TRUE	Product	CIM87::VerifySlotMap API check this value 0 = do not send on VerifySlotMap (old behavior) 1=send on VerifySlotMap when equipment based verification fails
CarrierCapacityDefault	FALSE	TRUE	Product	The default capacity of the carrier used in Bind, ProceedWithCarrier, CarrierNotification when the port specific capacity of the carrier is unknown.
CarrierCapacityDefault_[i]	FALSE	TRUE	Product	The default capacity of the ith carrier used in Bind, ProceedWithCarrier when the port specific capacity of the carrier is unknown.

12.2.4 Collection Events

Event Name	Private	User/ Product Handled	Developer Description

Event Name	**Private**	**User/ Product Handled**	**Developer Description**
AccessModeViolationClear	FALSE	Product	This event is triggered automatically by the product when the alarm is cleared. The user does not trigger this event explicitly.
AccessModeViolationSet	FALSE	Product	This event is triggered automatically by the product when the alarm is set. The user does not trigger this event explicitly.
AttemptToUseOutOfServiceLPClear	FALSE	Product	This event is triggered automatically by the product when the alarm is cleared. The user does not trigger this event explicitly.
AttemptToUseOutOfServiceLPSet	FALSE	Product	This event is triggered automatically by the product when the alarm is set. The user does not trigger this event explicitly.
BufferCapacityChanged	FALSE	Product	For internal buffer equipment only. This event is triggered by CIM87 in the following situations: 1. when a client application calls the function ICxE87CMS::CreateCarrier and the location is a carrier buffer. 2. When a client application calls the function ICxE87CMS.MoveCarrier or MoveCarrier2 if the carrier is moved in or out of a partition. 3. as a result of a successful Bind, ProceedWithCarrier, CarrierIn, CarrierNotification, or ReserveAtPort service.

Event Name	**Private**	**User/ Product Handled**	**Developer Description**
CarrierApproachingComplete	FALSE	User	Trigger this event in CIMConnect using one of the TriggerEvent functions based on some software or hardware indication that the completion is near. The interpretation of "approach completion" is somewhat subjective and left to the discretion of the equipment supplier and end user, but there are lots of guidelines in the E87 standard as quoted below. Cimetrix recommends defining a new Equipment Constant to make this event's behavior configurable to meet varying end-user requirements. 13.3.1 In some cases, for carrier transfer efficiency, the host needs to know carrier completion timing a little faster than actual. For example: •If the equipment is internal buffer type, QTAT carriers need to be moved out directly from internal FIMS to a load port to shorten moving out time. •If the equipment uses non-product carriers, such as dummy, they need to be changed before it becomes not reusable to prevent stopping the equipment operation. •If the equipment uses non-product carriers, such as test, reject, they need to be changed before it becomes empty or full to prevent stopping the equipment operation. 18.3.2 This event shall be generated when the access by the equipment to the carrier is approaching complete. How the timing of the event is determined shall be configurable. 18.3.3 Detailed definition of the event timing depends upon the type of usage of the carrier. Some examples of event timing for different types of usage are shown below. 18.3.3.1 PRODUCT — When remaining time until the carrier starts moving from internal FIMS to internal buffer reaches the configurable variable time (internal buffer equipment only). 18.3.3.2 DUMMY — When remaining times until substrates of the carrier becomes not reusable reaches the configurable variable times. 18.3.3.3 TEST — When remaining substrates until the carrier becomes empty reaches the configurable variable number. 18.3.3.4 REJECT — When remaining slots until the carrier becomes full reaches the configurable variable number. 18.3.4 Suppliers shall document the interpretation and the configurable variable(s) in the equipment specification document.

Event Name	Private	User/ Product Handled	Developer Description
CarrierClamped	FALSE	User	The user triggers this event in CIMConnect using one of the TriggerEvent functions based on some hardware or software indication that the carrier has been properly clamped.
CarrierClosed	FALSE	User	The user triggers this event in CIMConnect using one of the TriggerEvent functions based on some hardware or software indication that the carrier door has been properly closed.
CarrierDockFailureClear	FALSE	Product	This event is triggered automatically by the product when the alarm is cleared. The user does not trigger this event explicitly.
CarrierDockFailureSet	FALSE	Product	This event is triggered automatically by the product when the alarm is set. The user does not trigger this event explicitly.
CarrierIDReadFail	FALSE	Product	CIM87 triggers this event when a client application calls ICxE87CMS::CarrierAtPort with an empty carrierID string and readerVald is TRUE when the load port is in the NOT ASSOCIATED state.
CarrierLocationChanged	FALSE	Product	CIM87 triggers this event when a client application calls function ICxE87CMS::CarrierPlaced if the EPJ UseCarrierPlaced variable is set to 1 or anytime the client application calls function ICxE87CMS::MoveCarrier, ICxE87CMS::MoveCarrier2 or ICxE87CMS::MoveCarrier3.
CarrierOpened	FALSE	User	The user triggers this event in CIMConnect using one of the TriggerEvent functions based on some hardware or software indication that the carrier door has been properly opened.
CarrierOpenFailureClear	FALSE	Product	Do not trigger this event explicitly. It is triggered automatically when the alarm is cleared.
CarrierOpenFailureSet	FALSE	Product	Do not trigger this event explicitly. It is triggered automatically when the alarm is set.
CarrierPlacementErrorClear	FALSE	Product	Do not trigger this event explicitly. It is triggered automatically when the alarm is cleared.
CarrierPlacementErrorSet	FALSE	Product	Do not trigger this event explicitly. It is triggered automatically when the alarm is set.
CarrierPresenceErrorClear	FALSE	Product	Do not trigger this event explicitly. It is triggered automatically when the alarm is cleared.
CarrierPresenceErrorSet	FALSE	Product	Do not trigger this event explicitly. It is triggered automatically when the alarm is set.
CarrierProcessingStatusChanged	FALSE	Product	Do not trigger this event explicitly. It is triggered automatically when the alarm is set.
CarrierRemovalErrorClear	FALSE	Product	Do not trigger this event explicitly. It is triggered automatically when the alarm is cleared.
CarrierRemovalErrorSet	FALSE	Product	Do not trigger this event explicitly. It is triggered automatically when the alarm is set.

Event Name	**Private**	**User/ Product Handled**	**Developer Description**
CarrierUnclamped	FALSE	User	Trigger this event in CIMConnect using one of the TriggerEvent functions based on some hardware or software indication that the carrier has been properly unclamped.
CarrierVerificationFailureClear	FALSE	Product	Do not trigger this event explicitly. It is triggered automatically when the alarm is cleared.
CarrierVerificationFailureSet	FALSE	Product	Do not trigger this event explicitly. It is triggered automatically when the alarm is set.
DuplicateCarrierIDClear	FALSE	Product	Do not trigger this event explicitly. It is triggered automatically when the alarm is cleared.
DuplicateCarrierIDInProcess	FALSE	User	It is the client application's responsibility to determine if processing has begun on the first duplicate carrier and to trigger the DuplicateCarrierIDInProcess collection event if it has using one of the TriggerEvent functions. The Accessing Status of the carrier might be used to determine this (See CCxE87Carrier::get_AccessingStatus). It is also the equipment applications responsibility to cancel the duplicate carriers if appropriate.
DuplicateCarrierIDSet	FALSE	Product	Do not trigger this event explicitly. It is triggered automatically when the alarm is set.
E87Associated2Associated	FALSE	Product	Do not trigger this event explicitly; CIM87 triggers this event automatically.
E87Associated2NotAssociated	FALSE	Product	Do not trigger this event explicitly; CIM87 triggers this event automatically.
E87Auto2Manual	FALSE	Product	CIM87 triggers this event automatically when the host executes the ChangeAccess service or the client application calls function ICx87CMS::ChangeAccessMode or ICx87CMS::ChangeAccessMode2. The client application must implement either the ICxE87Callback7::ChangeAccessMode2 or ICxE87Callback::ChangeAccessMode callback and return S_OK.
E87Carrier2NoState	FALSE	Product	CIM87 triggers this event automatically when the client application calls function ICxE87MCS::CarrierDepartedPort, when the host issues a CancelBind or CancelCarrierNotificaiton, when the client application calls function ICxE87CMS::CancelBind or ICxE87CMS::CancelCarrierNotification. The client application must implement callbacks ICxE87Callback::CancelBind and ICxE87Callback::CancelCarrierNotification and return S_OK.

Event Name	**Private**	**User/ Product Handled**	**Developer Description**
E87CarrierSubstratesCreated	FALSE	Product	CIM87 triggers this event automatically when the client application calls ICxE87VerifySlotMap, all of the substrate object instances are created and when the slot map is successfully verified (equipment verification scenario). It is also called when the host issues a ProceedWithCarrier to perform host-based slot map verification. The client application must implement callback ICxE87Callback::ProceedWithCarrier and return S_OK.
E87IdNotRead2BypassIdVerificationOk	FALSE	Product	CIM87 triggers this event automatically when the client application calls CarrierAtPort with an empty carrierID string, readerValid = FALSE and the BypassReadID is TRUE. The user can either have a separate ByPassReadID_i for each load port or one ByPassReadID that applies to all load ports.
E87IdNotRead2BypassWaitingForHost	FALSE	Product	CIM87 triggers this event automatically when the client application calls CarrierAtPort with an empty carrierID string, readerValid = FALSE and the BypassReadID is FALSE. The user can either have a separate ByPassReadID_i for each load port or one ByPassReadID that applies to all load ports.
E87IdNotRead2IdVerificationOk	FALSE	Product	CIM87 triggers this event automatically when the client application calls CarrierAtPort with a carrier ID that matches
E87IdNotRead2WaitingForHost	FALSE	Product	CIM87 could not read the carrier id unsuccessfully (transition 7 in carrier state model). Carrier id status is set to WaitingForHost in CancelCarrier, CarrierAtPort, ProceedWithCarrier APIs.
E87InAccess2CarrierComplete	FALSE	User	CIM87 does not change the Accessing Status of a carrier, so it is up to the equipment application to change the accessing status. When the equipment is no longer accessing the carrier, and the carrier is ready to be moved out, AccessCarrier API is used to change the Accessing Status to either CARRIER_COMPLETE(the processing has completed normally).
E87InAccess2CarrierStopped	FALSE	User	CIM87 does not change the Accessing Status of a carrier, so it is up to the equipment application to change the accessing status. When the equipment is no longer accessing the carrier, and the carrier is ready to be moved out, AccessCarrier API is used to change the Accessing Status to either CARRIER_ STOPPED (the processing has completed abnormally for example aborted or stopped).
E87InService2OutOfService	FALSE	Product	CIM87::ChangeServiceStatus API sets the status depending on the ‘status’ attribute = 0 in S3F25 message. The operator also can directly call this API.
E87InService2TransferBlocked	FALSE	Product	In CIM87, ChangeLPTransferState, CreateCarrier, CreateLoadPort APIs may set the carrier status to transfer blocked and trigger E87InService2TransferBlocked event.

Event Name	**Private**	**User/ Product Handled**	**Developer Description**
E87InService2TransferReady	FALSE	Product	In CIM87, CreateLoadPort, Create2LocationLoadPort, ChangeLPTransferState APIs may set the carrier status to transfer ready and trigger E87InService2TransferReady event.
E87Manual2Auto	FALSE	Product	CIM87 triggers this event automatically when the host executes the ChangeAccess service (S3F27) or the client application calls function ICx87CMS::ChangeAccessMode or ICx87CMS::ChangeAccessMode2. The client application must implement either the ICxE87Callback7::ChangeAccessMode2 or ICxE87Callback::ChangeAccessMode callback and return S_OK.
E87NoState2Auto	FALSE	Product	In CIM87, CreateLoadPort and Create2LocationLoadPort APIs call ChangeLoadPortAssociatedState and trigger E87NoState2Auto event. The Load Port Access Mode State is restored to its last state. The "AccessMode_#" GEM status variable is used to restore the state. This variable is persistent so that the state can be restored during start up.
E87NoState2Carrier	FALSE	Product	In CIM87, CarrierAtPort calls ChangeLoadPortReservedState and the event is triggered in ChangeLoadPortReservedState in all of the following cases: 1) carrier id is read and it matches with the carrier id from the bind 2) carrier id is read and it does not match with the carrier id from carrier notification 3) carrier id read is successful Also, CreateCarrier API will trigger this event.
E87NoState2IdNotRead	FALSE	Product	When the BIND service has been accepted, CIM87creates Create a new carrier object and E87NoState2IdNotRead event is triggered.
E87NoState2IdVerficationOk	FALSE	Product	CIM87::ProceedWithCarrier triggers this event when a carrier is instantiated.
E87NoState2IdVerificationFail	FALSE	Product	CIM87::CancelCarrier triggers this event.
E87NoState2InService	FALSE	Product	In CIM87, ChangeLPTransferState, Create2LocationLoadPort, CreateCarrier, CreateLoadPort APIs trigger this event.
E87NoState2Manual	FALSE	Product	In CIM87, CreateLoadPort and Create2LocationLoadPort APIs call ChangeLoadPortAssociatedState and trigger E87NoState2Manual event. The Load Port Access Mode State is restored to its last state. The "AccessMode_#" GEM status variable is used to restore the state. This variable is persistent so that the state can be restored during start up.
E87NoState2NotAccessed	FALSE	Product	CIM87:: BIND API triggers this event when the bond service is accepted and a new carrier object is created.
E87NoState2NotAssociated	FALSE	Product	In CIM87, CreateLoadPort and Create2LocationLoadPort APIs trigger this event.

Event Name	**Private**	**User/ Product Handled**	**Developer Description**
E87NoState2OutOfService	FALSE	Product	In CIM87, ChangeLPTransferState, Create2LocationLoadPort, CreateLoadPort APIs trigger this event.
E87NoState2WaitingForHost	FALSE	Product	In CIM87, CarrierAtPort calls ChangeLoadPortReservedState and the event is triggered in all of the following cases: 1) carrier id is read and it matches with the carrier id from the bind 2) carrier id is read and it does not match with the carrier id from carrier notification 3) carrier id read is successful
E87NotAccess2InAccess	FALSE	Product	CIM87::AccessCarrier API triggers this event.
E87NotAssociated2Associated	FALSE	Product	In CIM87, CarrierAtPort API triggers this event all of the following cases: 1) carrier id is read and it matches with the carrier id from the bind 2) carrier id is read and it does not match with the carrier id from the bind 3) carrier id read fails CreateCarrier API aslo triggers this event.
E87NotReserved2Reserved	FALSE	Product	When the BIND service has been accepted, CIM87creates Create a new carrier object and E87NotReserved2Reserved event is triggered.
E87OutOfService2InService	FALSE	Product	In CIM87, ChangeLoadPortTransferState API triggers this event.
E87ReadyToLoad2TransferBlocked	FALSE	Product	Both MoveCarrier & TransferStart APIs in CIM87 call ChangeLoadPortReservedState and the event is triggered when the carrier is moved from a non-load port carrier location to load port location.
E87ReadyToUnload2TransferBlocked	FALSE	Product	During carrier removal, CIM87::TransferStart is called and the collection event E87ReadyToUnload2TransferBlocked will be triggered, and the state will be changed to LP_TRANSFER_BLOCKED.
E87Reserved2NotReserved	FALSE	Product	In CIM87, CarrierAtPort calls ChangeLoadPortReservedState and the event is triggered in ChangeLoadPortReservedState in all of the following cases: 4) carrier id is read and it matches with the carrier id from the bind 5) carrier id is read and it does not match with the carrier id from the bind 6) carrier id read fails
E87SlotMapNotRead2SlotMapVerificationOk	FALSE	Product	CIM87::VerifySlotMap changes the carrier SlotMap status to SM_VERIFICATION_OK and the collection event E87SlotMapNotRead2SlotMapVerificationOk is triggered.
E87SlotMapNotRead2WaitingForHost	FALSE	Product	CIM87::VerifySlotMap triggers the event during the slot map verification i.e. hosts based verification: the slotmap is read successfully and waiting for the host to verify the slotmap; equipment based verification: the slotmap is read successfully and the equipment fails to verify the slotmap.

Event Name	**Private**	**User/ Product Handled**	**Developer Description**
E87TransferBlocked2ReadytoLoad	FALSE	Product	CIM87:: ChangeLoadPortTransferState triggers this event.
E87TransferBlocked2ReadytoUnload	FALSE	Product	CIM87:: ChangeLoadPortTransferState triggers this event.
E87TransferBlocked2TransferReady	FALSE	Product	CIM87::ChangeLoadPortTransferState triggers this event when the transfer is unsuccessful and the carrier is not loaded or unloaded.
E87TransferReady2ReadyToLoad	FALSE	Product	CIM87::ChangeLoadPortTransferState triggers this event when carrier is not present.
E87TransferReady2ReadyToUnload	FALSE	Product	CIM87::ChangeLoadPortTransferState triggers this event when carrier is present.
E87WaitingforHost2IdVerificationFail	FALSE	Product	CIM87::CancelCarrier triggers this event upon receiving a carrier cancel command from the host or a direct API call from the equipment.
E87WaitingforHost2IdVerificationOk	FALSE	Product	CIM87::ProceedWithCarrier triggers this event upon receiving a proceed command from the host or a direct API call from the equipment.
E87WaitingForHost2SlotMapVerification Fail	FALSE	Product	CIM87::CancelCarrier and CIM87::CancelCarrierAtPort APIs trigger this event when the cancel carrier service is received from the host.
E87WaitingForHost2SlotMapVerification Ok	FALSE	Product	During the host based slot map verification, CIM87:: ProceedWithCarrier triggers this event
IBCarrierMoveFailureClear	FALSE	Product	Do not trigger this event explicitly. It is triggered automatically when the alarm is cleared.
IBCarrierMoveFailureSet	FALSE	Product	Do not trigger this event explicitly. It is triggered automatically when the alarm is set.
IDReaderAvailable	FALSE	User	Trigger this event in CIMConnect using one of the TriggerEvent functions based on some software or hardware indication that the id reader is available.
IDReaderUnavailable	FALSE	User	Trigger this event in CIMConnect using one of the TriggerEvent functions based on some software or hardware indication that the id reader is not available.
PIOFailureClear	FALSE	Product	Do not trigger this event explicitly. It is triggered automatically when the alarm is cleared.
PIOFailureSet	FALSE	Product	Do not trigger this event explicitly. It is triggered automatically when the alarm is set.
SlotMapReadFailedClear	FALSE	Product	Do not trigger this event explicitly. It is triggered automatically when the alarm is cleared.
SlotMapReadFailedSet	FALSE	Product	Do not trigger this event explicitly. It is triggered automatically when the alarm is set.
SlotMapVerificationFailedClear	FALSE	Product	Do not trigger this event explicitly. It is triggered automatically when the alarm is cleared.
SlotMapVerificationFailedSet	FALSE	Product	Do not trigger this event explicitly. It is triggered automatically when the alarm is set.
UnknownCarrierID	FALSE	Product	CIM87::CarrierAtPort API triggers this even automatically.
E87GeneralAlarmClear	FALSE	Product	Do not trigger this event explicitly. It is triggered automatically when the alarm is cleared.

Event Name	**Private**	**User/ Product Handled**	**Developer Description**
E87GeneralAlarmSet	FALSE	Product	Do not trigger this event explicitly. It is triggered automatically when the alarm is set.

12.2.5 Alarms

Alarm Name	User/ Product Handled	Developer Description
AccessModeViolation	User	The client application should set the AccessModeViolation alarm whenever an equipment or load port is set to AUTO and a carrier is manually delivered or when an equipment or load port is set to MANUAL and a carrier is automatically delivered. The meaning of this alarm is under ballot to be changed by the SEMI GEM 300 task force. Use the CIMConnect SetAlarm and ClearAlarm functions to set and clear the alarm. Modify the GEM 300 interface manual information to provide more details about when this alarm is set and cleared.
CarrierVerificationFailure	Both	When ICxE87CMS.CarrierAtPort is called, if SendCIDVerifyAlarm is set to 1, then the CarrierVerificationFailure alarm will be set by CIM87 if a carrier is placed that is not the Associated carrier. This alarm shall be set only when equipment-based Carrier ID verification fails. ICxE87Callback8.PortAlarmChanged is called when this load port-related alarm is set or cleared. Although the alarm is set by CIM87, the client application must clear the alarm. Use the CIMConnect ClearAlarm functions to clear the alarm. Modify the GEM 300 interface manual information to provide more details about when this alarm is cleared.
SlotMapReadFailed	User	The SlotMapReadFailed alarm should be set and cleared by the client application based on the results of the hardware. It can be set when the hardware detects a problem and reports that it has failed to read the slot map. It can be cleared when the administrator or user acknowledges the alarm or when the equipment detects a successful slot map read. Use the CIMConnect SetAlarm and ClearAlarm functions to set and clear the alarm. Modify the GEM 300 interface manual information to provide more details about when this alarm is set and cleared.
SlotMapVerificationFailed	Both	CIM87 triggers this error after the client application calls ICxE87CMS::VeritySlotMap during an equipment verification scenario and the slot map specified in VerifySlotMap does not match the expected slot map. ICxE87Callback8.PortAlarmChanged is called when this load port-related alarm is set or cleared. Although the alarm is set by CIM87, the client application must clear the alarm. Use the CIMConnect ClearAlarm functions to clear the alarm. Modify the GEM 300 interface manual information to provide more details about when this alarm is cleared.
AttemptToUseOutOfService LP	Both	If a load port is in the OUT OF SERVICE state, then CIM87 will set the AttemptToUseOutOfServiceLP alarm in situations where there is an attempt to use the Bind, ProceedWithCarrier, ReserveAtPort or CarrierOut services. ICxE87Callback8.PortAlarmChanged is called when this load port-related alarm is set or cleared.

Alarm Name	User/ Product Handled	Developer Description
		Although the alarm is set by CIM87, the client application must clear the alarm. Use the CIMConnect ClearAlarm functions to clear the alarm. Modify the GEM 300 interface manual information to provide more details about when this alarm is cleared.
CarrierPresenceError	User	Should be set whenever a carrier is at a location but is not expected to be there. Use the CIMConnect SetAlarm and ClearAlarm functions to set and clear the alarm. Modify the GEM 300 interface manual information to provide more details about when this alarm is set and cleared.
CarrierPlacementError	User	Should be set whenever a carrier is at the loadport, but is not correctly placed. Use the CIMConnect SetAlarm and ClearAlarm functions to set and clear the alarm. Modify the GEM 300 interface manual information to provide more details about when this alarm is set and cleared.
CarrierDockFailure	User	Should be set whenever a carrier cannot be docked or undocked. Use the CIMConnect SetAlarm and ClearAlarm functions to set and clear the alarm. Modify the GEM 300 interface manual information to provide more details about when this alarm is set and cleared.
CarrierOpenFailure	User	Should be set whenever a carrier door cannot be opened or closed. Use the CIMConnect SetAlarm and ClearAlarm functions to set and clear the alarm. Modify the GEM 300 interface manual information to provide more details about when this alarm is set and cleared.
DuplicateCarrierID	Both	If the equipment receives a carrier with a CarrierID that is the same as that of another carrier present at the equipment, the following rules shall apply: 1. Duplicate CarrierID alarm shall be set to notify the host. 2. The second carrier with a CarrierID shall not be processed. 3. If processing on the first carrier with the CarrierID has not begun, it should not be processed. 4. If processing on the first carrier has begun a Duplicate Carrier ID In Process event shall be issued to notify the host. CIM87 will set this alarm in applicable situatuations when the client application calls API function ICxE87CMS::CarrierAtPort or when there is an attempt to use the Bind, ProceedWithCarrier, CancelCarrier, or CarrierNotification services. Although the alarm is set by CIM87, the client application must clear the alarm. Use the CIMConnect ClearAlarm functions to clear the alarm. Modify the GEM 300 interface manual information to provide more details about when this alarm is cleared.
CarrierRemovalError	User	Should be set whenever there is a failure while attempting to remove (unload) a carrier. Use the CIMConnect SetAlarm and ClearAlarm functions to set and clear the alarm. Modify the GEM 300 interface manual information to provide more details about when this alarm is set and cleared.
PIOFailure	User	Should be set whenever there is a failure in the E84 (parallel input/output) PIO

Alarm Name	User/ Product Handled	Developer Description
		interface. This includes when the communication link fails, an AMHS requests a load transfer and the port is not READY_TO_LOAD, or an AMHS requests an unload and the port is not READY_TO_UNLOAD. Use the CIMConnect SetAlarm and ClearAlarm functions to set and clear the alarm. Modify the GEM 300 interface manual information to provide more details about when this alarm is set and cleared.
IBCarrierMoveFailure	User	This alarm applies only to internal buffer equipment only. Should be set whenever a failure occurs while attempting to move a carrier to/from carrier buffer locations. . Use the CIMConnect SetAlarm and ClearAlarm functions to set and clear the alarm. Modify the GEM 300 interface manual information to provide more details about when this alarm is set and cleared.
E87GeneralAlarm	Product	This alarm is set when calls are made to ProceedWithCarrier or CancelCarrier when the E87 object is in an invalid state for those methods.

12.3 SEMI Standard ID (E90)/Cimetrix Module ID (CIM90)

12.3.1 Data Variables

Data Variable Name	Private	Persistent	User/ Product Handled	Developer Description
AcquiredID	FALSE	FALSE	Product	For equipment with substrate ID readers only. Contains the ID read from the substrate using the substrate ID reader. It is an empty string before the substrate is read. The attribute should be updated as soon the Substrate ID has been successfully read. The value is updated when the CIM300 application calls CCxE90ST::NotifySubstrateRead. Empty string if the read failed.
AcquiredIDList	FALSE	FALSE	User	This is not handled by CIM300.
BatchLocID	FALSE	FALSE	Product	Required only for batch equipment. The Batch Location associated with the last event where the Substrate was instantiated.
BatchLocState	FALSE	FALSE	Product	Required only for batch equipment.
BatchSubstIDMap	FALSE	FALSE	Product	Required only for batch equipment.
LotID	FALSE	FALSE	Product	
SubstBatchLocID	FALSE	FALSE	Product	Required only for batch equipment.
SubstBatchLocIDList	FALSE	FALSE	Product	Required only for batch equipment.
SubstDestination	FALSE	FALSE	Product	
SubstDestinationList	FALSE	FALSE	User	This is not handled by CIM300.
SubstHistory	FALSE	FALSE	Product	
SubstHistoryList	FALSE	FALSE	Product	

Data Variable Name	Private	Persistent	User/ Product Handled	Developer Description
SubstID	FALSE	FALSE	Product	
SubstIDList	FALSE	FALSE	Product	
SubstIDStatus	FALSE	FALSE	Product	
SubstIDStatusList	FALSE	FALSE	Product	
SubstLocID	FALSE	FALSE	Product	If for batch process equipment, this data is empty while the substrate belongs to a batch.
SubstLocIDList	FALSE	FALSE	Product	
SubstLocState	FALSE	FALSE	Product	
SubstLocStateList	FALSE	FALSE	Product	An ordered list of substrate location states.
SubstLocSubstID	FALSE	FALSE	Product	CIM300 maintains this value automatically and ensures it is set correctly when reported with E90 Substrate Location events. This DV should be used instead of the SubstID DV for Substrate Location events. It will contain a substrate id value when location is OCCUPIED and Blank "" when location is UNOCCUPIED.
SubstLotID	FALSE	FALSE	Product	
SubstLotIDList	FALSE	FALSE	Product	
SubstMtrlStatus	FALSE	FALSE	User	The enumeration is user defined. Document the meaning of the values used in the GEM interface manual. Set the value using CCxE90ST::SetSubstrateMaterialStatus.
SubstMtrlStatusList	FALSE	FALSE	User	
SubstPosInBatch	FALSE	FALSE	Product	Required only for batch equipment.
SubstPosInBatchList	FALSE	FALSE	Product	Required only for batch equipment.
SubstProcState	FALSE	FALSE	Product	Updated using CCxE90ST::ChangeSubstrateState, CCxE90ST::ChangeSubstrateState2, or CCxE90ST::SetSubstrateTransportState.
SubstProcStateList	FALSE	FALSE	Product	
SubstratePreviousIDStatus	FALSE	FALSE	Product	
SubstSource	FALSE	FALSE	Product	
SubstSourceList	FALSE	FALSE	Product	
SubstState	FALSE	FALSE	Product	Updated using CCxE90ST::ChangeSubstrateState, CCxE90ST::ChangeSubstrateState2, or CCxE90ST::SetSubstrateProcessingState.
SubstStateList	FALSE	FALSE	Product	
SubstSubstLocID	FALSE	FALSE	Product	Updated using CCxE90ST::ChangeSubstrateState or CCxE90ST::ChangeSubstrateState2.
SubstType	FALSE	FALSE	Product	By default, the type will be setWAFER. Use the CCxSubstrate::currentEquipmentType property to change it to a different type.
SubstTypeList	FALSE	FALSE	Product	

Data Variable Name	Private	Persistent	User/ Product Handled	Developer Description
SubstUsage	FALSE	FALSE	Product	By default, the usage will be seuPRODUCT. Use the CCxSubstrate:: currentEquipmentUsage property to change it to a different usage. Or to change the default usage, create a variable named “UsageDefault” of type ASCII and set its value to “PRODUCT”, “TEST”, “FILLER”, or “CLEANING”. ASCII string “DUMMY” is also mapped to the seuFILLER enumeration.
SubstUsageList	FALSE	FALSE	Product	

12.3.2 Status Variables

Status Variable Name	Private	Persistent	User/ Product Handled	Developer Description
BatchLocID_[i]	FALSE	FALSE	Product	Required only for batch equipment. The batch location ID for location [i]. The value is updated when the CIM300 application calls API CCxE90ST::CreateBatch or CCxE90ST::DestroyBatch. The number of BatchLocID_[i] variables in the EPJ is determined by the maximum number of concurrent batches possible in the equipment.
BatchLocState_[i]	FALSE	FALSE	Product	Required only for batch equipment. The state of the batch location [i]. The value is updated when the CIM300 application calls API CCxE90ST::ChangeBatchState or CCxE90ST::ChangeBatchStateAndHistory. The number of BatchLocState_[i] variables in the EPJ is determined by the maximum number of concurrent batches possible in the equipment.
BatchSubstIDMap_[i]	FALSE	TRUE	Product	Required only for batch equipment. The state of the batch location [i]. The value is updated when the CIM300 application calls API CCxE90ST::CreateBatch or CCxE90ST::DestroyBatch. The number of BatchSubstIDMap_[i] variables in the EPJ is determined by the maximum number of concurrent batches possible in the equipment.
SubstLocID_[i]	FALSE	FALSE	Product	Create a SubstLocID_ status variable for each static substrate location within the tool, and 25 for each carrier that can be at the tool at one time (SubstLocID_1, SubstLocID_2, etc.). The [i] locations are allocated based on the order the application calls CCxE90ST::AddSubstrateLocation, at which time the SubstLocID_[i] value is set. Document the assignment of each SubstLocState_[i] in the GEM interface manual.

Status Variable Name	**Private**	**Persistent**	**User/ Product Handled**	**Developer Description**
SubstLocState_[i]	FALSE	FALSE	Product	Create a SubstLocState _ status variable for each static substrate location within the tool, and 25 for each carrier that can be at the tool at one time (SubstLocState _1, SubstLocState _2, etc.). The [i] locations are allocated based on the order the application calls CCxE90ST::AddSubstrateLocation. The value is updated when the application calls CCxE90ST::ChangeSubstrateState or CCxE90ST::ChangeSubstrateState2. Document the assignment of each SubstLocState_[i] in the GEM interface manual.
SubstrLocSubstrID_[i]	FALSE	TRUE	Product	Create a SubstrLocSubstrID _ status variable for each static substrate location within the tool, and 25 for each carrier that can be at the tool at one time (SubstrLocSubstrID _1, SubstrLocSubstrID _2, etc.) . The [i] locations are allocated based on the order the application calls CCxE90ST::AddSubstrateLocation. The value is updated when the application calls CCxE90ST::ChangeSubstrateState or CCxE90ST::ChangeSubstrateState2. Document the assignment of each SubstrLocSubstrID _[i] in the GEM interface manual. CIM300 maintains this value automatically. The value is empty (zero-length) when the substrate location is unoccupied, and CIM300 reports a zero-length value if included in the OCCUPIED-to-UNOCCUPIED event report for that location. If the E90ComplianceLevel is set for backward compatibility (a value less than 200411), the value is not cleared until after the event is sent, and any event report including this variable contains the substrate ID that just left the location.

12.3.3 Equipment Constants

Equipment Constant Name	**Private**	**Persistent**	**User/ Product Handled**	**Developer Description**
SubstLocCount	TRUE	TRUE	Product	This setting is used to configure the maximum number of substrate locations that are supported by the equipment. This includes the substrate locations in the equipment and in the carriers. If SubstLocCount is not specified in the EPJ file, the default is 256. If this configuration option is specified in the EPJ file, the maximum number will be determined by this value. The EPJ file must also contain the corresponding number of subscripted E90 variables SubstLocState_[i], SubstLocID_[i] and SubstrLocSubstrID_[i] and equipment constant SubstLocDisableEvents_[i] where i=1 to the value of SubstLocCount.
BatchLocDisableEvents_[i]	FALSE	TRUE	Product	Required only for batch equipment.

Equipment Constant Name	**Private**	**Persistent**	**User/ Product Handled**	**Developer Description**
				When set to TRUE, the transitions for the ith batch location do not generate events to the host. TRUE Disable, FALSE Enable
E90ComplianceLevel	FALSE	TRUE	Product	As of SR9, Cimetrix added a compliance level variable for the 300mm standards. If you are using CIM300 SR9 or later, you must add these compliance level variables to your epj file. The default for each of these variables is to use the latest compliance that is currently supported by CIM300. For examples of the default settings, see the C:\Program Files\Cimetrix\Comm Products\support\CIM300\E90ST.epj file. The value is a numeric value with the 4-digit year and 2-digit month of the revision supported (YYYYMM). For the SEMI E90-1105 revision, set the value to 200511. If you need to disable the new features that are associated with later releases of the standards (2004, 2005, etc.), you must set these variables to older standard levels. If E90ComplianceLevel is set to a value of 200307 or less, CIM300 operates at the 2003 07 (July) compliance level. 0 is an acceptable value. In the case of E90ComplianceLevel and if you are using a compliance level of 200511 (or later), add all of the E90 variables that are associated with this release (batch variables are still optional).
E90UseSlotmapOnly	FALSE	TRUE	Product	Required only for batch equipment. This value is used when calling CCxE90ST::AddCarrierToBatch or CCxE90ST::AddCarrierToBatch2.
LocationNoStateEvents	FALSE	TRUE	Product	This value is used when substrates are created and deleted, either directly or indirectly through CIM87.
SubstLocDisableEvents_[i]	FALSE	TRUE	Product	
SubstrateBatchEventsStyle	FALSE	TRUE	Product	Required only for batch equipment. For backward compatibility with CIM300 versions before SR9, set the value to 0.
SubstrateReaderEnabled	FALSE	TRUE	Both	The SEMI E90-1105 revision added support for substrate ID readers. CIM300 includes several recent changes to support these features, including a new state model, new supporting variables and services, and events and other items associated with the new state substrate object state model. Allow this to be set TRUE only if the E90ComplianceLevel setting in the EPJ file is set to 200411 or later. If 0, do not call CCxE90ST::NotifySubstrateRead or expect substrate ID verification callbacks.
UsageDefault	TRUE	FALSE	Product	See DV SubstUsage.
SuppressGroupSubLocEven ts	FALSE	TRUE	Product	Required only for batch equipment.
CarrierTxferGroupEvents	FALSE	TRUE	Product	The data type for the EC variable “CarrierTxferGroupEvents”

Equipment Constant Name	**Private**	**Persistent**	**User/ Product Handled**	**Developer Description**
				was changed from a Boolean (Bo) to a Byte (U1), which allows for a third possible value which permits only substrate object events (andnot substrate location events) to be grouped during a carrier transfer. The SEMI E90 standard is not clear if it is expected that substrate location object state change events should be grouped along with substrate object state change events during a carrier transfer when carrier transfer group events are turned on. Because of this lack of clarity, Cimetrix added a new value (2) for the CarrierTxferGroupEvents EC variable. The new value indicates that only substrate events should be grouped. The CarrierTxferGroupEvents EC now permits three values: 0 = no grouping, 1 = group both substrate object and substrate location object events, and 2 = group substrate events only.
SubstObjType	TRUE	FALSE	Product	Defines the ObjType for substrates. By default, this is SUBSTRATE. This would be changed only when replacing a previous GEM 300 implementation where the ObjType for substrates was different than SUBSTRATE. 902000=SubstObjType,The E90 Substrate ObjType value.,,EC,A,,,1,,,A SUBSTRATE,
SubstLocObjType	TRUE	FALSE	Product	Defines the ObjType for substrate locations. By default, this is SUBSTLOC. This would be changed only when replacing a previous GEM 300 implementation where the ObjType for substrate locations s was different than SUBSTRATE. 902001=SubstLocObjType,The E90 Substrate Location ObjType value.,,EC,A,,,1,,,A SUBSTLOC,
BatchLocObjType	TRUE	FALSE	Product	Required only for batch equipment. Defines the ObjType for batch locations. By default, this is BATCHLOC. This would be changed only when replacing a previous GEM 300 implementation where the ObjType for batch locations s was different than BATCHLOC. 902002=BatchLocObjType,The E90 Batch Location ObjType value.,,EC,A,,,1,,,A BATCHLOC,

12.3.4 Collection Events

Event Name	**Private**	**User/ Product Handled**	**Developer Description**
E90_BatchLoc_Occupied2Unoccupied	FALSE	Product	Required only for batch equipment. Use API CCxE90ST::ChangeBatchState or CCxE90ST::ChangeBatchStateAndHistory to trigger this event when a batch changes location. The previous batch location becomes unoccupied.
E90_BatchLoc_Unoccupied2Occupied	FALSE	Product	Required only for batch equipment. Use API CCxE90ST::ChangeBatchState or

Event Name	Private	User/ Product Handled	Developer Description
			CCxE90ST::ChangeBatchStateAndHistory to trigger this event when a batch changes location. The new batch location becomes occupied.
E90_ID_NotConfirmed2Confirmed	FALSE	Product	When the application calls CCxE90ST::NotifySubstrateRead with argument goodRead true, CIM90 triggers this event if the acquired substrate ID matches the substrate ID from the ContentMap.
E90_ID_NotConfirmed2WaitingForHost	FALSE	Product	When the application calls CCxE90ST::NotifySubstrateRead with argument goodRead false, CIM90 triggers this event.
E90_ID_NotConfirmed2WaitingForHost2	FALSE	Product	When the application calls CCxE90ST::NotifySubstrateRead with argument goodRead true, CIM90 triggers this event if the acquired substrate ID does not match the substrate ID from the ContentMap.
E90_ID_WaitingForHost2ConfirmationFailed	FALSE	Product	CIM90 triggers this event when the host sends a CancelSubstrate service while in a Waiting for Host state.
E90_ID_WaitingForHost2Confirmed	FALSE	Product	CIM90 triggers this event when the host sends a ProceedWithSubstrate service while in a Waiting for Host state.
E90_Loc_Occupied2Unoccupied	FALSE	Product	When the application calls CCxE90ST::ChangeSubstrateState or CCxE90ST::ChangeSubstrateState2, CIM90 triggers this event for the substrate location which has become unoccupied.
E90_Loc_Unoccupied2Occupied	FALSE	Product	When the application calls CCxE90ST::ChangeSubstrateState or CCxE90ST::ChangeSubstrateState2, CIM90 triggers this event for the substrate location which has become occupied.
E90_Subst_AnyState2Extinction	FALSE	Product	When the application or CIM87 calls CCxE90ST::RemoveSubstrate, CIM90 triggers this event if the substrate’s state is not stsAT_DESTINATION. Typically CIM87 handles this when a carrier is removed.
E90_Subst_AtDestination2AtSource	FALSE	Product	When the application calls CCxE90ST::ChangeSubstrateState, CCxE90ST::ChangeSubstrateState2, or CCxE90ST::SetSubstrateTransportState to change the transport state from stsAT_DESTINATION to stsAT_SOURCE, CIM90 triggers this event.
E90_Subst_AtDestination2AtWork	FALSE	Product	When the application calls CCxE90ST::ChangeSubstrateState, CCxE90ST::ChangeSubstrateState2, or CCxE90ST::SetSubstrateTransportState to change the transport state from stsAT_DESTINATION to stsAT_WORK, CIM90 triggers this event.
E90_Subst_AtDestination2Extinction	FALSE	Product	When the application or CIM87 calls CCxE90ST::RemoveSubstrate, CIM90 triggers this event if the substrate’s state is stsAT_DESTINATION. Typically CIM87 handles this when a carrier is removed.
E90_Subst_AtSource2AtWork	FALSE	Product	When the application calls CCxE90ST::ChangeSubstrateState, CCxE90ST::ChangeSubstrateState2, or

Event Name	**Private**	**User/ Product Handled**	**Developer Description**
			CCxE90ST::SetSubstrateTransportState to change the transport state from stsAT_SOURCE to stsAT_WORK, CIM90 triggers this event.
E90_Subst_AtWork2AtDestination	FALSE	Product	When the application calls CCxE90ST::ChangeSubstrateState, CCxE90ST::ChangeSubstrateState2, or CCxE90ST::SetSubstrateTransportState to change the transport state from stsAT_WORK to stsAT_DESTINATION, CIM90 triggers this event.
E90_Subst_AtWork2AtSource	FALSE	Product	When the application calls CCxE90ST::ChangeSubstrateState, CCxE90ST::ChangeSubstrateState2, or CCxE90ST::SetSubstrateTransportState to change the transport state from stsAT_WORK to stsAT_SOURCE, CIM90 triggers this event.
E90_Subst_AtWork2AtWork	FALSE	Product	When the application calls CCxE90ST::ChangeSubstrateState, CCxE90ST::ChangeSubstrateState2, or CCxE90ST::SetSubstrateTransportState to change the transport state from stsAT_WORK to stsAT_WORK, CIM90 triggers this event.
E90_Subst_InProcess2NeedsProcessing	FALSE	Product	When the application calls CCxE90ST::ChangeSubstrateState, CCxE90ST::ChangeSubstrateState2, or CCxE90ST::SetSubstrateProcessingState to change the processing state from spsIN_PROCESS to spsNEEDS_PROCESSING, CIM90 triggers this event.
E90_Subst_InProcess2ProcessingComplete	FALSE	Product	When the application calls CCxE90ST::ChangeSubstrateState, CCxE90ST::ChangeSubstrateState2, or CCxE90ST::SetSubstrateProcessingState to change the processing state from spsIN_PROCESS to spsPROCESSED, spsABORTED, spsSTOPPED, spsREJECTED, spsLOST or spsSKIPPED, CIM90 triggers this event.
E90_Subst_NeedsProcessing2InProcess	FALSE	Product	When the application calls CCxE90ST::ChangeSubstrateState, CCxE90ST::ChangeSubstrateState2, or CCxE90ST::SetSubstrateProcessingState to change the processing state from spsNEEDS_PROCESSING to spsIN_PROCESS, CIM90 triggers this event.
E90_Subst_NeedsProcessing2ProcessingComplete	FALSE	Product	When the application calls CCxE90ST::ChangeSubstrateState, CCxE90ST::ChangeSubstrateState2, or CCxE90ST::SetSubstrateProcessingState to change the processing state from spsNEEDS_PROCESSING to spsPROCESSED, spsABORTED, spsSTOPPED, spsREJECTED, spsLOST or spsSKIPPED, CIM90 triggers this event.
E90_Subst_NoState2AtSource	FALSE	Product	When the application or CIM87 calls

Event Name	Private	User/ Product Handled	Developer Description
			CCxE90ST::RegisterSubstrate or CCxE90ST::RegisterSubstrate2, CIM90 triggers this event. Typically CIM87 handles this when a carrier arrives.
E90_Subst_NoState2NeedsProcessing	FALSE	Product	When the application or CIM87 calls CCxE90ST::RegisterSubstrate or CCxE90ST::RegisterSubstrate2, CIM90 triggers this event. Typically CIM87 handles this when a carrier arrives.
NoState2Occupied	FALSE	Product	When the application or CIM87 calls CCxE90ST::AddSubstrateLocation with a substrate ID specified, CIM90 triggers this event. Typically CIM87 handles this when a carrier arrives. The application must handle this for static substrate locations that are not part of the carrier.
NoState2Unoccupied	FALSE	Product	When the application or CIM87 calls CCxE90ST::AddSubstrateLocation with an empty substrate ID argument, CIM90 triggers this event. Typically CIM87 handles this when a carrier arrives. The application must handle this for static substrate locations that are not part of the carrier.
Occupied2NoState	FALSE	Product	When the application or CIM87 calls CCxE90ST::RemoveSubstrateLocation and the location is occupied, CIM90 triggers this event.
SubstrateIDReaderAvailable	FALSE	Product	Use API CCxE90ST::SubstrateReaderAvailable to trigger this event when the Substrate ID reader becomes available after being unavailable.
SubstrateIDReaderUnavailable	FALSE	Product	Use API CCxE90ST::SubstrateReaderAvailable to trigger this event when the Substrate ID reader becomes unavailable after being available.
Unoccupied2NoState	FALSE	Product	When the application or CIM87 calls CCxE90ST::RemoveSubstrateLocation and the location is unoccupied, CIM90 triggers this event.

12.4 SEMI Standard ID (E94)/Cimetrix Module ID (CIM94)

12.4.1 Data Variables

Data Variable Name	Private	Persistent	User/ Product Handled	Developer Description
CtrlJobID	FALSE	FALSE	Product	The object ID (object identifier) of a Control Job, which is available to use in related events and event reports. Conforms to ObjID in SEMI E39.1, Section 6. Format: 1–80 characters
DataCollectionPlan	FALSE	FALSE	User	The identifier for a data collection plan that is used during execution of the control job. It's a name given by the host to associate data collection activities to a specific control job. In general, it provides a

Data Variable Name	Private	Persistent	User/ Product Handled	Developer Description
				way for the equipment to inform and coordinate with the host to receive data collection requests. A DataCollectionPlan is generic and will be applied to many control jobs. The variable holds no significance for the equipment. It is just a label that the equipment reports back when requested by the host.
CtrlJobState	FALSE	FALSE	Product	The state of the Control Job. E94CJState is enumerated as follows: -1 – cjUNKNOWN – (No State) 0 – cjQUEUED 1 – cjSELECTED 2 – cjWAITINGFORSTART 3 – cjEXECUTING 4 – cjPAUSED 5 – cjCOMPLETED Range: U1 0-U1 255
CJCreateERRTEXT	FALSE	FALSE	User	ERRTEXT for application rejection of control job creation.
CJCreateERRCOD E	FALSE	FALSE	User	ERRCODE for application rejection of control job creation. Default: U4 0
CurrentPRJob	FALSE	FALSE	Product	Holds the identifiers of any process job in the active state (for example, in the Executing, Stopping, Aborting or Pause states). It's a list of PRJobIDs (see SEMI E40).
CarrierInputSpec	FALSE	FALSE	User	A list of carrierIDs for materials that will be used by the ControlJob. An empty list is allowed. CJCreate used to fail when there was a mismatch between CarrierInputSpec and the PRMtlNameList of the associated Process Jobs. For SR10 and later versions, CIM300 now resets the parent Control Job attirbute of the associated Process Job and updates the underlying Process Jobs to release the parent attribute, allowing the Process Job to be included in a new Control Job. Format: L,n n = number of input carriers 1. <CARRIERID1> … n. <CARRIERIDn>.
MtrlOutSpec	FALSE	FALSE	User	A list structure that maps material from source to destination after processing. For uni-carrier operation or if CarrierInputSpec is an empty list.
MtrlOutbyStatus	FALSE	FALSE	User	A list structure that maps locations or Carriers where processed material is placed based on the material status.
PauseEvent	FALSE	FALSE	Product	Identifier of a list of event IDs (CEIDs) on which the Control Job must PAUSE.
ProcessingCtrlSpec	FALSE	FALSE	User	ProcessingCtrlSpec attribute.

Data Variable Name	**Private**	**Persistent**	**User/ Product Handled**	**Developer Description**
ProcessOrderMgmt	FALSE	FALSE	Product	An enumeration that defines the method for the order in which process jobs are initiated. The standard defines three enumerations, but other enumerations are possible for some equipment. 1 ARRIVAL 2 OPTIMIZE 3 LIST
StartMethod	FALSE	FALSE	Product	A logical flag that determines if the ControlJob can start automatically. A user start can come through either the host connection or the operator console. TRUE – Auto FALSE – UserStart
PRJobStatusList	FALSE	FALSE	Product	A list of all process jobs managed by this Control Job and their associated status, where n is the number of process jobs that are defined in the control job. CIM300 maintains this value automatically.

12.4.2 Status Variables

Status Variable Name	**Private**	**Persistent**	**User/ Product Handled**	**Developer Description**
QueuedCJobs	FALSE	FALSE	Product	This is an ordered list of control jobs that are currently in the Queue. The first job in the list is the job at the head of the Queue. Each list item is a control job identifier. This value is used to query the names of the control jobs that are currently residing in the queue.
QueueAvailableSpace	FALSE	FALSE	Product	Indicates the number of jobs that the Queue can accept. This value cannot be negative. When it is 0, it indicates that the queue is full. Format: 5() (U1, U2, U4, U8)

12.4.3 Equipment Constants

Equipment Constant Name	**Private**	**Persistent**	**User/ Product Handled**	**Developer Description**
SetUpName	FALSE	TRUE	User	A Host sets this to define the operational condition of the equipment. If the equipment is manipulated locally, this variable should be "unknown." The equipment uses a zero-length string to indicate "unknown." Otherwise, it returns the value that is set by the host (when requested). Format: 1-80 characters
CJWaitFor1DayAfterComplete	FALSE	TRUE	Product	Specifies whether to delete a Control Job immediately after it has completed or delete it after it has persisted for at least one day. 1=CJ will be removed after it has persisted for at least one day in COMPLETED state, 0=CJ will be removed immediately
CJCheckSlotMapStatusMaterial	FALSE	TRUE	Product	1=CJ will wait for Slot Map Status of Carrier(s) to be verified, 0=will not check.
CheckMtrlOutSpecFmt	FALSE	TRUE	Product	If set to 1 and the format of MtrlOutSpec is invalid, the request to create a Control Job is rejected.
CheckMtrlOutByStatusFmt	FALSE	TRUE	Product	If set to 1 and the format of MtrlOutByStatus attribute is invalid, the request to create a Control Job is rejected.
CJCreateReturn0LenObjSpec	FALSE	TRUE	Product	1=Return a zero length OBJSPEC in the S14F10 message 0=Behave as normal (default)
ClearAttrListOnError	FALSE	TRUE	Product	1=If the CJCreate callback returns an error, clear the attributes list 0=Do not clear the attribute list
CJCarrierInputMatch	FALSE	TRUE	Product	Permits rejection of a control job create command when the CarrierInputSpec does not match the PRMtlNameList of associated Process Jobs. When the variable is set to 1, this new feature is enabled. If the EPJ variable is not included in the EPJ or is set to 0, the default behavior is invoked and CIM300 does not compare the variables

Equipment Constant Name	Private	Persistent	User/ Product Handled	Developer Description
				between the Process Job(s) and the Control Job. 0=Default 1=Reject CJ on mismatch
E94ComplianceLevel	TRUE	TRUE	Product	As of SR9, Cimetrix added a compliance level variable for the 300mm standards. If you are using CIM300 SR9 or later, you must add these compliance level variables to your epj file. The default for each of these variables is to use the latest compliance that is currently supported by CIM300. For examples of the default settings, see the C:\Program Files\Cimetrix\Comm Products\support\CIM300\E94CJM.epj file. The value is a numeric value with the 4-digit year and 2-digit month of the revision supported. For the SEMI E94-0306 revision, set the value to 200603. If you need to disable the new features that are associated with later releases of the standards (2004, 2005, etc.), you must set these variables to older standard levels. If E94ComplianceLevel is set to a value of 200307 or less, CIM300 operates at the 2003 07 (July) compliance level. 0 is an acceptable value. In the case of E94ComplianceLevel, if you are using a compliance level of 200603 (or later), ensure that the E40ComplianceLevel (for Process Jobs) is set to 200507 or later. If you set E94ComplianceLevel to at least 200603, E40ComplianceLevel to at least 200507, and E40ProcessJobPersistence to at least 1, your jobs won't disappear when calling CCxE40PJM::PRJobComplete and CCxE94CJM::ProcessJobComplete. Instead, the jobs are deleted when the material is removed. Format: YYYYMM
E94ObjType	TRUE	TRUE	Product	E94 Control Job Object Type attribute value. “CONTROLJOB”

12.4.4 Collection Events

Event Name	Private	User/ Product Handled	Developer Description
E94_NoState2Queued	FALSE	Product	Control Job transitioned from “No State” to QUEUED state -- Tool receives “Create” command. Create ControlJob and put it at the tail of the queue

Event Name	**Private**	**User/ Product Handled**	**Developer Description**
E94_Queued2NoState	FALSE	Product	Control Job transitioned from the QUEUED state to “No State” – Too receives “Cancel”, “Abort”, or “Stop” command. The job is de-queued and terminated.
E94_Queued2Selected	FALSE	Product	Control Job transitioned from the QUEUED state to the SELECTED state – Tool selects and de-queues the job at the head of the queue.
E94_Selected2Queued	FALSE	Product	Control Job transitioned from the SELECTED state to the QUEUED state – Tool receives de-select command and materials for the control job have not arrived yet. The job is de-selected and the job is moved to the head of the queue.
E94_Selected2Executing	FALSE	Product	Control Job transitioned from the SELECTED state to the EXECUTING state -- Material for the first process job arrives or in the case where the first (or only) process job does not require material, this transition shall be taken as soon as the processing resource for that process job becomes available. “StartMethod” attribute in the ControlJob is set for Auto.
E94_Selected2WaitingForStart	FALSE	Product	Control Job transitioned from the SELECTED state to the WAITINGFORSTART state -- Material for the first process job arrives or in the case where the first (or only) process job does not require material, this transition shall be taken as soon as the processing resource for that process job becomes available. “StartMethod” attribute in the ControlJob is set to UserStart.
E94_WaitingForStart2Executing	FALSE	Product	Control Job transitioned from the WAITINGFORSTART state to the EXECUTING state – User START command received.
E94_Executing2Paused	FALSE	Product	Control Job transitioned from the EXECUTING state to the PAUSED state – Tool receives a pause command.
E94_Paused2Executing	FALSE	Product	Control Job transitioned from the PAUSED state to the EXECUTING state – Tool receives a RESUME command and process jobs begin executing.
E94_Executing2Completed	FALSE	Product	Control Job transitioned from the EXECUTING state to the COMPLETED state – All process jobs for the ControlJob have completed.
E94_Active2Completed_Stopped	FALSE	Product	Control Job transitioned from any “Active” state to the STOPPED state – The equipment receved a CJStop command and all process jobs under the ControlJob that were ACTIVE at the time the command was received have entered the POST ACTIVE state.
E94_Active2Completed_Aborted	FALSE	Product	Control Job transitioned from any “Active” state to the ABORTED state -- The equipment receved a CJAbort command and all process jobs under the ControlJob that were ACTIVE at the time the command was received have entered the POST ACTIVE state.
E94_Completed2NoState	FALSE	Product	Control Job transitioned from the COMPLETED state to “No State” – Any required material has been removed and the ControlJob is deleted.
SubstrateDefinitionNotAccessible Set	FALSE	Product	This event is triggered automatically by the product when the alarm is cleared. The user does not trigger this event explicitly
SubstrateDefinitionNotAccessible	FALSE	Product	This event is triggered automatically by the product when the

Event Name	Private	User/ Product Handled	Developer Description
Clear			alarm is cleared. The user does not trigger this event explicitly
SubstrateDefinitionUnknownSet	FALSE	Product	This event is triggered automatically by the product when the alarm is cleared. The user does not trigger this event explicitly
SubstrateDefinitionUnknownClear	FALSE	Product	This event is triggered automatically by the product when the alarm is cleared. The user does not trigger this event explicitly

12.4.5 Alarms

Alarm Name	User/ Product Handled	Developer Description
SubstrateDefinitionNotAccessible	User	For equipment supporting Material Redirection Mode, when the MtrlOutSpec is not empty, the client application should set the SubstrateDefinitionNotAccessible alarm whenever the equipment is ready to return material to its destination, but the destination location is not accessible or is occupied. The client application should clear this alarm when a DestinationMap referring to an accessible destination for the material has been provided by the host or operator intervention. Use the CIMConnect SetAlarm and ClearAlarm functions to set and clear the alarm. Modify the GEM 300 interface manual information to provide more details about when this alarm is set and cleared.
SubstrateDefinitionUnknown	User	For equipment supporting Material Redirection Mode, if the equipment supports incomplete MtrlOutSpecs, and the DestinationMap is empty upon completion of processing, the client application should set the SubstrateDefinitionUnknown alarm if the equipment is unable to determine a destination location from the MtrlOutSpec. The client application should clear this alarm when the material has been successfully moved to its destination location. Additional information is available in section 7.3.3.2. Use the CIMConnect SetAlarm and ClearAlarm functions to set and clear the alarm. Modify the GEM 300 interface manual information to provide more details about when this alarm is set and cleared.

12.5 SEMI Standard ID (E116)/Cimetrix Module ID (CIM116)

12.5.1 Data Variables

Data Variable Name	Private	Persistent	User/ Product Handled	Developer Description
BlockedReason	FALSE	FALSE	Product	An enumeration that identifies the most recent blocked condition that initiated the transition to the BLOCKED state for the related EPT module or equipment. 0 Not Blocked 1 Unknown

Data Variable Name	**Private**	**Persistent**	**User/ Product Handled**	**Developer Description**
				2 Safety Threshold 3 Error Condition 4 Parametric Exception 5 Aborting, Aborted 6 Pausing, Paused 7 Reserved 8 Reserved 9 Reserved
BlockedReasonText	FALSE	FALSE	Product	A description of the reason why the transition was made to the BLOCKED state for the EPT module or equipment. It can provide further details to BlockedReason. This attribute must be defined for both equipment and module EPTTracker objects. For the Equipment, a BlockedReason value is determined by the blocked module. Its usage is equipment defined. For example, "Abort – Host-requested Abort," "Fault – Reticle Stage Error," "Pause – Host-requested Pause." Format: 0-80 charcters
EPTElementName	FALSE	FALSE	Product	A human-understandable name for this equipment or module, which must be unique among all objects of EPTTracker type. Usage is equipment defined. The user's application defines the EPT element names when creating the EPT modules. CIM300 maintains this variable automatically and ensures that the DV has the correct value when triggering E116 events. Examples include: EQUIPMENT CHAMBER A ROBOT ARM Format: 0–80 characters
EPTElementType	FALSE	FALSE	Product	An enumeration of numeric codes that indicate whether this EPTTracker object refers to the equipment or an EPT module. 0 = Equipment 1 = Production EPT Module 2 = EFEM/LoadPort EPT Module
EPTState	FALSE	FALSE	Product	An enumeration of the new (resulting) EPT state of the EPT module or equipment at the end of an EPT state transition. 0 = Idle 1 = Busy 2 = Blocked 3 = NoState
EPTStateTime	FALSE	FALSE	Product	This is the time period (in seconds) between the entry into the previous EPT state and the entry into the current EPT State.
PreviousEPTState	FALSE	FALSE	Product	An enumeration of the previous EPT state of this equipment or EPT

Data Variable Name	**Private**	**Persistent**	**User/ Product Handled**	**Developer Description**
				module, prior to entering the current EPTState. 0 Idle 1 Busy 2 Blocked 3 No State
PreviousTaskName	FALSE	FALSE	Product	The name of the EPT Task that was previously running on this EPT module, prior to starting the current EPT Task. This attribute must be defined for both equipment and module EPTTracker objects. For the Equipment, the value of the PreviousTaskName attribute can be left blank (an empty string). Its usage is equipment defined. Examples include: "Loading," "Etching," and "Pumping Down." Format: 0–80 characters
PreviousTaskType	FALSE	FALSE	Product	An enumeration of the type of the EPT Task that was previously running on this EPT module, prior to starting the current EPT Task. This attribute must be defined for both equipment and module EPTTracker objects. For the Equipment, the value of the PreviousTaskType attribute can be set to 0. 0 No Task 1 Unspecified 2 Process -- adding value (for example, exposing) 3 Support -- incapable of adding value (for example, handling/transport) 4 Equipment Maintenance (for example, equipment-initiated clean cycle) 5 Equipment Diagnostics (for example, equipment-initiated health check) 6 Waiting (for example, chamber waiting for a robot to remove a substrate)
TaskName	FALSE	FALSE	Product	The name of the EPT Task that is currently running on this EPT module. This attribute must be defined for both equipment and module EPTTracker objects. For the Equipment, the value of the TaskName attribute can be blank (empty string). This is defined by the equipment supplier. Examples include “Loading," “Etching,” and “Pumping Down.” Format: 0-80 characters
TaskType	FALSE	FALSE	Product	An enumeration of the type of EPT Task that is currently running on this EPT module. This attribute must be defined for both equipment and module EPTTracker objects. For the Equipment, the value of the TaskType attribute can be 0. 0 No Task 1 Unspecified 2 Process -- adding value (for example, exposing) 3 Support -- incapable of adding value (for example, handling/transport) 4 Equipment Maintenance (for example, equipment-initiated clean

Data Variable Name	Private	Persistent	User/ Product Handled	Developer Description
				cycle) 5 Equipment Diagnostics (for example, equipment-initiated health check) 6 Waiting (for example, chamber waiting for a robot to remove a substrate)

12.5.2 Status Variables

Status Variable Name	Private	Persistent	User/ Product Handled	Developer Description
BlockedEFEMModuleList	FALSE	FALSE	Product	
BlockedProductionModuleList	FALSE	FALSE	Product	
BlockedReason_[i]	FALSE	FALSE	Product	An enumeration that identifies the most recent blocked condition that initiated the transition to the BLOCKED state for the ith module or equipment. 0 Not Blocked 1 Unknown 2 Safety Threshold 3 Error Condition 4 Parametric Exception 5 Aborting, Aborted 6 Pausing, Paused 7 Reserved 8 Reserved 9 Reserved
BlockedReasonText_[i]	FALSE	FALSE	Product	A description of the reason why the transition was made to the BLOCKED state for the EPT module or equipment. It can provide further details to BlockedReason. This attribute must be defined for both equipment and module EPTTracker objects. For the Equipment, a BlockedReason value is determined by the blocked module. Its usage is equipment defined. For example, “Abort – Host-requested Abort,” “Fault – Reticle Stage Error,” “Pause – Host-requested Pause.” Format: 0-80 charcters
BusyEFEMModuleList	FALSE	FALSE	Product	A list of Busy EFEM and Loadport EPT Module object IDs for this equipment.
BusyProductionModuleList	FALSE	FALSE	Product	A list of all BUSY Production EPT object IDs for this equipment.
DisableEventOnTransition	FALSE	TRUE	Product	A list of zero to nine EPT State Model transitions for which event reporting to the host are disabled. The list can

Status Variable Name	**Private**	**Persistent**	**User/ Product Handled**	**Developer Description**
				be empty. Range: List of zero to nine integers in the range 1–9, in order, with no repeats Format: L,n 1. 51 (U1) <Transition Number> 2. 51 (U1) <Transition Number> … n. 51 (U1) <Transition Number>
DisableEventOnTransition_[i]	FALSE	TRUE	Product	
EFEMModuleList	FALSE	FALSE	Product	The list of EPT EFEM/Loadport Module ObjIDs for the equipment.
EPTElementName_[i]	FALSE	TRUE	Product	A human-understandable name for this equipment or module, which must be unique among all objects of EPTTracker type. Usage is equipment defined. The user's application defines the EPT element names when creating the EPT modules. CIM300 maintains this variable automatically. Examples include: EQUIPMENT CHAMBER A ROBOT ARM Format: 0-80 characters
EPTState_[i]	FALSE	FALSE	Product	An enumeration of the new (resulting) EPT state of the ith EPT module or equipment at the end of an EPT state transition. 0 = Idle 1 = Busy 2 = Blocked 3 = NoState
EPTStateTime_[i]	FALSE	FALSE	Product	This is the time period (in seconds) between the entry into the previous EPT state and the entry into the current EPT State for the ith module or equipment.
IdleEFEMModuleList	FALSE	FALSE	Product	A list of Idle EFEM/Loadport EPT Module ObjIDs for this equipment.
IdleProductionModuleList	FALSE	FALSE	Product	A list of Idle Production EPT module ObjIDs for this equipment.
ModuleList	FALSE	FALSE	Product	A list of all EPT Module ObjIDs for this equipment.
NumBlocked	TRUE	FALSE	Product	This variable keeps track of the number of EPT modules in the BLOCKED state. It is no longer required by SEMI E116. This should be configured as a private variable in the EPJ file (not available to the host) for support of SEMI

Status Variable Name	**Private**	**Persistent**	**User/ Product Handled**	**Developer Description**
				E116=0705 or later.
NumBusy	TRUE	FALSE	Product	This variable keeps track of the number of EPT modules in the BUSY state. It is no longer required by SEMI E116. This should be configured as a private variable in the EPJ file (not available to the host) for support of SEMI E116=0705 or later.
NumEFEMLoadportBlocked	TRUE	FALSE	Product	This variable keeps track of the number of EFEM/Loadport modules in the BLOCKED state. It is no longer required by SEMI E116. This should be configured as a private variable in the EPJ file (not available to the host) for support of SEMI E116=0705 or later.
NumEFEMLoadportBusy	TRUE	FALSE	Product	This variable keeps track of the number of EFEM/Loadport modules in the BUSY state. It is no longer required by SEMI E116. This should be configured as a private variable in the EPJ file (not available to the host) for support of SEMI E116=0705 or later.
NumEFEMLoadportIdle	TRUE	FALSE	Product	This variable keeps track of the number of EFEM/Loadport modules in the IDLE state. It is no longer required by SEMI E116. This should be configured as a private variable in the EPJ file (not available to the host) for support of SEMI E116=0705 or later.
NumEFEMLoadportModules	TRUE	FALSE	Product	This variable keeps track of the number of EFEM/Loadport modules. It is no longer required by SEMI E116. This should be configured as a private variable in the EPJ file (not available to the host) for support of SEMI E116=0705 or later.
NumIdle	TRUE	FALSE	Product	This variable keeps track of the number of EPT modules in the IDLE state. It is no longer required by SEMI E116. This should be configured as a private variable in the EPJ file (not available to the host) for support of SEMI E116=0705 or later.
NumModules	TRUE	FALSE	Product	This variable keeps track of the number of EPT modules. It is no longer required by SEMI E116. This should be configured as a private variable in the EPJ file (not available to the host) for support of SEMI E116=0705 or later.
NumProductionBlocked	TRUE	FALSE	Product	This variable keeps track of the number of EPT Process modules in the BLOCKED state. It is no longer required

Status Variable Name	Private	Persistent	User/ Product Handled	Developer Description
				by SEMI E116. This should be configured as a private variable in the EPJ file (not available to the host) for support of SEMI E116=0705 or later.
NumProductionBusy	TRUE	FALSE	Product	This variable keeps track of the number of EPT Process modules in the BUSY state. It is no longer required by SEMI E116. This should be configured as a private variable in the EPJ file (not available to the host) for support of SEMI E116=0705 or later.
NumProductionIdle	TRUE	FALSE	Product	This variable keeps track of the number of EPT modules in the IDLE state. It is no longer required by SEMI E116. This should be configured as a private variable in the EPJ file (not available to the host) for support of SEMI E116=0705 or later.
NumProductionModules	TRUE	FALSE	Product	This variable keeps track of the number of EPT Process modules. It is no longer required by SEMI E116. This should be configured as a private variable in the EPJ file (not available to the host) for support of SEMI E116=0705 or later.
PreviousEPTState_[i]	FALSE	FALSE	Product	An enumeration of the previous EPT state of the ith equipment or EPT module. 0 Idle 1 Busy 2 Blocked 3 No State
PreviousTaskName_[i]	FALSE	FALSE	Product	The name of the EPT Task that was previously running on the ith EPT module, prior to starting the current EPT Task. This attribute must be defined for both equipment and module EPTTracker objects. For the Equipment, the value of the PreviousTaskName attribute can be left blank (an empty string). Its usage is equipment defined. Examples include: "Loading," "Etching," and "Pumping Down." Format: 0–80 characters
PreviousTaskType_[i]	FALSE	FALSE	Product	An enumeration of the type of the EPT Task that was previously running on the ith EPT module, prior to starting the current EPT Task. This attribute must be defined for both equipment and module EPTTracker objects. For the Equipment, the value of the PreviousTaskType attribute can be set to 0. 0 No Task 1 Unspecified

Status Variable Name	**Private**	**Persistent**	**User/ Product Handled**	**Developer Description**
				2 Process -- adding value (for example, exposing) 3 Support -- incapable of adding value (for example, handling/transport) 4 Equipment Maintenance (for example, equipment-initiated clean cycle) 5 Equipment Diagnostics (for example, equipment-initiated health check) 6 Waiting (for example, chamber waiting for a robot to remove a substrate)
ProductionModuleList	FALSE	FALSE	Product	A list of text ObjIDs that represent all the Production EPT Modules for this equipment.
SubstCount	TRUE	TRUE	Product	This variable keeps track of the number of substrates that are processed for all EPT modules. It is no longer required by SEMI E116. This should be configured as a private variable in the EPJ file (not available to the host) for support of SEMI E116=0705 or later.
SubstCount_[i]	TRUE	TRUE	Product	This variable keeps track of the number of substrates that are processed for a specific EPT module. It is no longer required by SEMI E116. This should be configured as a private variable in the EPJ file (not available to the host) for support of SEMI E116=0705 or later.
SubstCountReset	TRUE	TRUE	Product	This variable indicates the time at which the SubstCount DV was reset. Note however, that it is no longer required by SEMI E116. This should be configured as a private variable in the EPJ file (not available to the host) for support of SEMI E116=0705 or later. Format: YYYYMMDDHHMMSScc
SubstCountReset_[i]	TRUE	TRUE	Product	This variable indicates the time at which a specific SubstCount_[i] value was reset. Note however, that it is no longer required by SEMI E116. This should be configured as a private variable in the EPJ file (not available to the host) for support of SEMI E116=0705 or later. Format: YYYYMMDDHHMMSScc
TaskName_[i]	FALSE	FALSE	Product	The name of the EPT Task that is currently running on the ith EPT module. This attribute must be defined for both equipment and module EPTTracker objects. For the Equipment, the value of the TaskName attribute can be blank (empty string).

Status Variable Name	Private	Persistent	User/ Product Handled	Developer Description
				This is defined by the equipment supplier. Examples include "Loading," "Etching," and "Pumping Down." Format: 0-80 characters
TaskType_[i]	FALSE	FALSE	Product	An enumeration of the type of EPT Task that is currently running on the ith EPT module. This attribute must be defined for both equipment and module EPTTracker objects. For the Equipment, the value of the TaskType attribute can be 0. 0 No Task 1 Unspecified 2 Process -- adding value (for example, exposing) 3 Support -- incapable of adding value (for example, handling/transport) 4 Equipment Maintenance (for example, equipment-initiated clean cycle) 5 Equipment Diagnostics (for example, equipment-initiated health check) 6 Waiting (for example, chamber waiting for a robot to remove a substrate)

12.5.3 Equipment Constants

EC Name	Private	Persistent	User/ Product Handled	Developer Description
E116ComplianceLevel	TRUE	TRUE	Product	As of SR9, Cimetrix added a compliance level variable for the 300mm standards. If you are using CIM300 SR9 or later, you must add these compliance level variables to your epj file. The default for each of these variables is to use the latest compliance that is currently supported by CIM300. The value is a numeric value with the 4-digit year and 2-digit month of the revision supported (YYYYMM). For the SEMI E116-0705 revision, you would set the value to 200507. If you need to disable the new features that are associated with later releases of the standards (2004, 2005, etc.), you must set these variables to older standard levels. If E116ComplianceLevel is set to a value of 200407 or less, CIM300 operates at the 2004 07 (July) compliance level. 0 is an acceptable value. For examples of the default settings, see the C:\Program Files\Cimetrix\Comm Products\support\CIM300\E116EPT.epj file.

12.5.4 Collection Events

Event Name	Private	User/ Product Handled	Developer Description

EPTStateChange	FALSE	Product	CIM116:: TriggerTransition API triggers the EPTStateChange event for the given transition.
EPTStateChange_[i]	FALSE	Product	CIM116:: TriggerTransition API triggers the EPTStateChange event for the given transition.

12.6 SEMI Standard ID (E157)/Cimetrix Module ID (CIM157)

12.6.1 Data Variables

Data Variable Name	**Private**	**Persistent**	**User/ Product Handled**	**Developer Description**
RCID	FALSE	FALSE	Product	
RecId	FALSE	FALSE	Product	
ModuleId	FALSE	FALSE	Product	
SubstrateId	FALSE	FALSE	Product	
SubstateIdList	FALSE	FALSE	Product	
ProcessJobId	FALSE	FALSE	Product	
ProcessJobIdList	FALSE	FALSE	Product	
RecipeParameters	FALSE	FALSE	Product	
StepId	FALSE	FALSE	Product	
StepCount	FALSE	FALSE	Product	

12.6.2 Status Variables

Status Variable Name	**Private**	**Persistent**	**User/ Product Handled**	**Developer Description**
ModuleId_[i]	FALSE	FALSE	Product	The value is assigned for the specified index when the application calls AddModule.
ModuleState_[i]	FALSE	FALSE	Product	Updated for the specified index when the application calls MPTExecutionStarted, MPTExecutionStarted2, MPTExecutionCompleted, MPTStepStarted, or MPTStepCompleted.
ModulePreviousState_[i]	FALSE	FALSE	Product	Updated for the specified index when the application calls MPTExecutionStarted, MPTExecutionStarted2, MPTExecutionCompleted, MPTStepStarted, or MPTStepCompleted. The previous value of MPTState _[i] is assigned to PreviousMPTState_[i].

12.6.3 Equipment Constants

Equipment Constant Name	**Private**	**Persistent**	**User/ Product Handled**	**Developer Description**

12.6.4 Collection Events

Event Name	Private	User/ Product Handled	Developer Description
ExecutionStarted_[i]	FALSE	Product	Call MPTExecutionStarted or MPTExecutionStarted2, depending on whether there is a single substrate or multiple substrates.
ExecutionCompleted_[i]	FALSE	Product	Call MPTExecutionCompleted with the successful flag set to VARIANT_TRUE.
ExecutionFailed_[i]	FALSE	Product	Call MPTExecutionCompleted with the successful flag set to VARIANT_FALSE.
StepStarted_[i]	FALSE	Product	Call MPTStepStarted.
StepCompleted_[i]	FALSE	Product	Call MPTStepCompleted with the successful flag set to VARIANT_TRUE.
StepFailed_[i]	FALSE	Product	Call MPTStepCompleted with the successful flag set to VARIANT_FALSE.