



Features & Benefits of

SECS/GEM

*A guide to some of the main features and benefits
of GEM – SEMI Standard E30*

Contents

3	Chapter 1: Introduction
6	Chapter 2: GEM Collection Events
13	Chapter 3: Data Polling
18	Chapter 4: GEM Factory Application Support
21	Chapter 5: Alarms
23	Chapter 6: Recipe Management
26	Chapter 7: Documentation
32	Chapter 8: Equipment Terminal Services
34	Chapter 9: User Interface
35	Chapter 10: GEM Message Spooling Capabilities
37	Chapter 11: Protocol Layer
42	Chapter 12: Message Logging
45	Chapter 13: GEM Control State
48	Chapter 14: Summary

Chapter 1

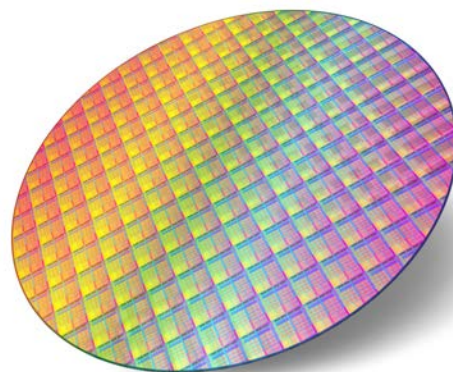
Introduction

SECS/GEM refers to a set of SEMI standards that govern the communication between manufacturing equipment and factory host computer systems. The Message layer standard, SEMI E5 SECS-II, defines a generic message structure and a library consisting of many standardized messages. The Protocol layer standard, SEMI E37 High-Speed Message Service (HSMS), defines a binary structure to transfer SECS-II messages using TCP/IP. SEMI E30 GEM, defines a minimum set of requirements, additional (optional) capabilities, use cases and user scenarios for a subset of SECS-II messages.

SECS/GEM is implemented on an equipment and is used by the factory to implement command and control functions. Since it is an industry standard, any SECS/GEM-compliant host software can communicate with any SECS/GEM-compliant equipment. When fully implemented on the equipment, the standards enable factory software to completely control and monitor the equipment by means of its SECS/GEM interface. These standards provide numerous benefits to both equipment manufacturers and factories. Several of these benefits are highlighted in this book.

SECS/GEM Reduces Equipment Integration Costs

Factories are typically owned and operated by multinational enterprises which purchase equipment from a variety of equipment manufacturers. Even though the control software is different on every equipment, the factory is required to integrate the equipment to operate in harmony. While it is possible to independently integrate each equipment with custom software, this is not cost or time effective.



The situation is similar for equipment manufacturers, who sell their products to diverse factories across the globe. Data collection and application software at every factory are different. The equipment manufacturer is required to help the factory integrate the purchased equipment. While it is possible to develop a custom integration solution for each factory, this is again not cost effective. Every time a factory asks for custom integration features, these costs get passed on to the factory itself.

Custom software, whether developed by the equipment manufacturer or the factory, is expensive to create and maintain, and tends to be of lower quality than desired. By contrast, the SECS/GEM standards define how to create a standardized interface on any manufacturing equipment.

Equipment manufacturers benefit by developing one interface for all their customers. Factories benefit by reusing the same integration software for all their purchased equipment. Reuse of this software and technology both by the factory and equipment manufacturer raises the software quality, reduces costs and allows for more functionality. The equipment manufacturer and factory alike can invest not only in the minimum features required, they can also implement advanced functionality that is otherwise unaffordable. If they only have to support SECS/GEM, then equipment manufacturers can publish more data and support more advanced control. In turn, factories can then use the additional data to improve product quality and productivity.

SECS/GEM Is Applicable to All Manufacturing Equipment

Because SECS/GEM is divided into Fundamental Requirements and Additional Capabilities, it can be implemented on any manufacturing equipment, regardless of size and complexity. Additional Capabilities are optional because they are not always needed. For example, some equipment do not have recipes and therefore do not need to implement the Recipe Management Additional Capability.

SECS/GEM also scales well with the magnitude of an equipment's data. For example, a very simple equipment or device might publish 10 different collection events, whereas a complex equipment might publish 5000 different collection events; yet both can use the same SECS/GEM technology.

Innumerable Applications Can Be Supported Using a SECS/GEM Interface

Everything that happens on an equipment can be tracked. Any remote control features and system configuration can be supported. The more data that is published by an equipment, the more software applications a factory can implement. A SECS/GEM interface makes it possible to implement applications for statistical process control, troubleshooting, predictive maintenance, feedforward/feedback process control, equipment utilization, material tracking, recipe validation and many more. Such applications often reduce the need for an operator interface on the equipment, thereby reducing the number of operators in the factory. Recipe management allows factories to minimize scrap. For example, use the SECS/GEM interface to store golden recipes in a central location and also to ensure that the correct recipe is used on the material.

SECS/GEM Uses Network Bandwidth Very Efficiently

There are several features that make SECS/GEM naturally efficient. First, every SECS/GEM interface acts as a message broker. Because the broker runs on the equipment, unsubscribed data is not published on the network. For host software to receive alarm, collection event, or trace data messages, it must first subscribe. Since subscriptions for each alarm, collection event, and trace data are managed separately, the equipment can implement a single SECS/GEM interface that publishes all alarms, collection events and trace data requested by all factory applications without wasting network bandwidth with unnecessary data. Moreover, when the host subscribes for trace data, it specifies the data collection rate, making SECS/GEM much more efficient and useful than protocols that publish data at a hard-coded rate.

Additionally, all SECS/GEM messages are always transmitted in an efficient binary format. This uses much less bandwidth than protocols that transmit in ASCII format. Despite using a binary format, SECS/GEM messages are also easily converted to and from a standardized XML notation.

SECS/GEM Enjoys Enormous Industry Support

SECS/GEM has been the backbone of factory/equipment communication and control systems for years in the semiconductor industry. This means that all semiconductor manufacturing today completely relies on SECS/GEM communication. 300mm semiconductor factories have been running with full automation based on SECS/GEM communication since the late 90s—large companies like TSMC, Samsung, Micron, Intel, Toshiba and many others utilize SECS/GEM 24/7 in every factory. Other industries like Flat Panel Display, High-Brightness LED and Photovoltaic have also officially adopted SECS/GEM because they recognized how SECS/GEM features can be applied to any manufacturing equipment to support mission-critical applications.

SECS/GEM Is Self-Describing

Although the standard requires GEM documentation to be provided with the equipment, SECS/GEM supports multiple approaches for host software to automatically adapt to an equipment's SECS/GEM interface. There are messages for the host software to ask for the list of available alarms, status variables, equipment constants, and, for newer implementations, a list of available collection events and data variables. These messages make SECS/GEM plug-and-play. Additionally, the equipment manufacturer can provide a standardized XML file that provides a full description of the SECS/GEM interface and its features.

Summary

These are just some of the many benefits of using SECS/GEM technology, both for factories and equipment manufacturers. SECS/GEM is proven technology that is available today. Read on to discover these and other features and benefits in detail.

Chapter 2

GEM Collection Events

To start off our in-depth coverage on SECS/GEM (or just GEM) features and benefits, let's begin with an explanation of one of the GEM standard's key features called Collection Events. We'll start with an explanation as to how they work, then move to why they are so effective for collecting data from manufacturing equipment.

-What are collection events?

The two words in the name "collection event" are descriptive.

SECS/GEM Reduces Equipment Integration Costs

As denoted by the word "event" a collection event is a notification. Its purpose is to notify the host when something of interest happens at the equipment. The "host" is the factory client software that connects to the equipment's GEM interface. For example, collection events can report when material arrived, a consumable is running low, a hardware problem occurred, a camera inspected the material, the material is ready to be removed, a chamber reached the target vacuum pressure, processing completion, etc. The equipment can use the collection event feature to report when anything of interest happens. Whoever makes the GEM interface determines exactly what collection events are available to the host; therefore, the set of available collection events is different from equipment type to equipment type.

As denoted by the word "collection", collection events are also capable of publishing data along with the collection event message. It is a very efficient form of data collection, asynchronously providing information as it becomes available. For example, a collection event that reports when material arrives might also report the arriving material's barcode and location.

There are three types of data in a GEM interface; information about the collection event (called data variables), status information (called status variables) and equipment settings (called equipment constants). Whoever makes the GEM interface determines exactly what information will be available for each collection event. So the set of available information for collection events is different from equipment type to equipment type. And the available data is only sent if the host sets up the reports.

What are collection events?



The two words in the name "collection event" are descriptive.

A Little Analogy

As an analogy, think of the factory as a boss and the equipment they purchase as employees. There are many different styles of managing, just like there are different types of factories and styles for running a factory. You don't want to be forced to run your factory just like someone else's factory. You want to run it your way.



Additionally, each employee is unique and needs unique level of attention. And each employee is doing unique things. Generally speaking, all managers want to know basic information about employees and what their employees are doing. They want to know when the employee starts a project and when they finish a project. Some employees are very productive even with minimal oversight and reporting. Some employees need extensive oversight and reporting. GEM allow the factory to deal with each equipment uniquely. Specifically, GEM collection events give the equipment a way to report on what it is doing.

The host has to set up the rules for the reporting and adapt the rules appropriately. For example, sometimes a manager does not care when the employee goes to the bathroom. For certain employees, the manager might want to know. In a GEM interface, the host can choose which notifications occur and which do not.

Sometimes a manager just needs to be told when the employee does things like when employee arrives, departs, goes on break, and come off break. Sometimes a manager needs more details, like what project did you finish, how long did it take, the key results of the project. Similarly, GEM allows the host to track not when things happen, but to also provide details about the activity. GEM reports meet this need very effectively.

Why do you need this feature?

The short answer is that collection events allow you to track what the equipment is doing in real time. If a factory wants to any degree of Smart Manufacturing or just wants to improve productivity, then one of the first things needed is the ability to track what the equipment is doing. Collection events provide this. You can track equipment utilization, material movement, processing milestones, count cycles of activity for predictive maintenance, consumable usage, and anything else related to the published collection events. The applications for such information are endless.

Sometimes collection events are also used to implement scenarios where the equipment needs information from the host before proceeding or permission to proceed. A collection event tells the host when the equipment is ready for the host instructions or permission.

How does the collection event notification work?

An equipment's GEM interface can publish many different collection events. The host will not typically want to be notified of all of them at once and it does not have to. Collection events use a publish/subscribe design pattern in two ways.



Basic Publish/Subscribe Notification

The host subscribes to specific collection events to receive notification when they occur. The subscription allows a host to enable or disable the reporting of each collection event available in the GEM interface. The equipment publishes the collection events as they happen.

Event Report Publish/Subscribe Data Collection

By default, a collection event message will not include any data. A subscription also allows the host to decide what data to include in each enabled collection event's message. The host defines reports and links the reports to collection events; thereby subscribing to the data. Each collection event can have a different report. Reports can also be shared across multiple collection events. A report can include any data variables associated with the collection event, any status variables and any equipment constants. The equipment publishes the collection event with the requested data.

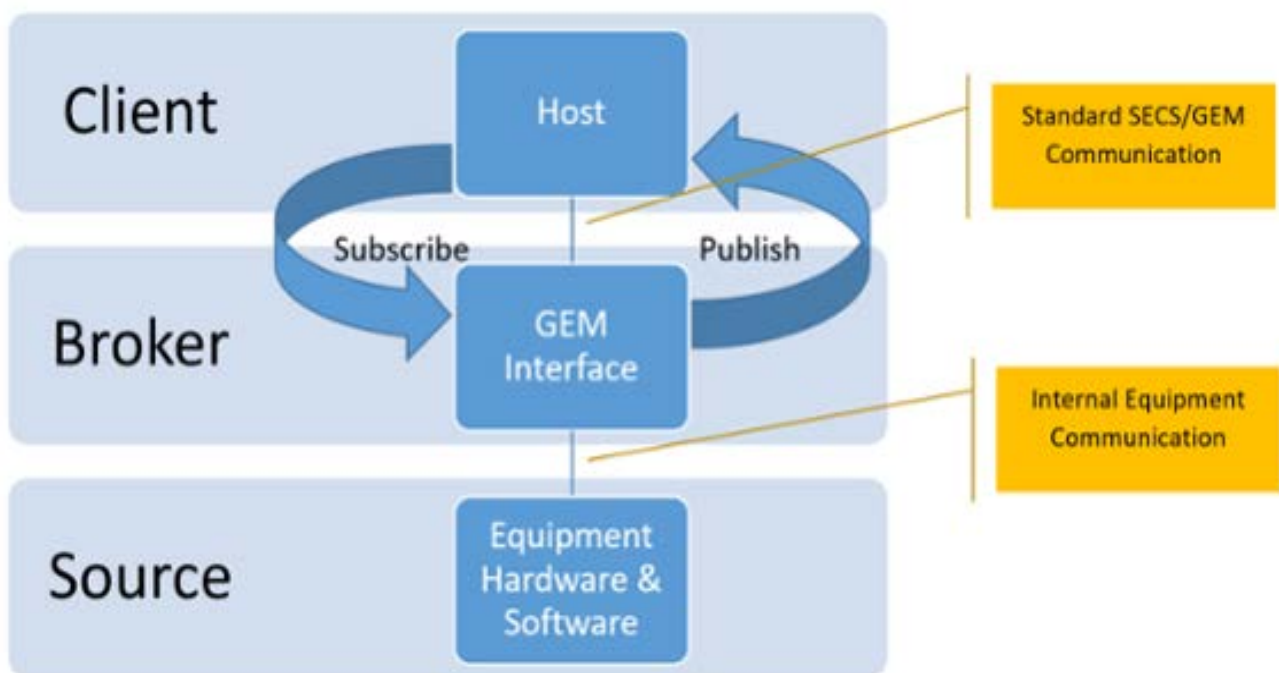
Identification

Each collection event published by the equipment has a unique ID number for identification. The host software uses the ID number when enabling or disabling a collection event. The equipment uses the ID number when the collection event message is sent. Each available data variable, status variable and equipment constant also has a unique ID number. When the host defines a report, it assigned the report a unique ID number.

Broker

The broker to handle all collection event publication/subscription is built into the equipment's GEM interface. It is part of the equipment system. Communication between the host (a client) and GEM interface is standardized using SECS/GEM communication. Communication between the GEM interface and the rest of the equipment hardware and software (the source of the equipment collection events and data) can be any appropriate technology and does not matter as long as the GEM interface functions properly and performs sufficiently well.

This means that messages are only sent from the equipment to the host when the host has subscribed. Having the broker as part of the equipment and GEM interface makes the GEM interface very efficient and use much less bandwidth than protocols that use an external broker where all messages and data have to be sent to a broker all the time.



Persistence

The collection event subscriptions are persisted in a GEM interface. So, if the host disconnects and reconnects, or if the equipment is restarted, the GEM interface will remember the setup of all subscriptions.

Which messages are used?

Here is a summary of each of the primary messages related to collection events. Note that the "S" identifies the "stream" and "F" identifies the "function". Together, a stream and function number uniquely identify a message.

Message ID	Direction	Description
S2F37	Host -> Equipment	<p>Enable or disable reporting for a set of collection events.</p> <p>An empty list will enable or disable the reporting for all collection events. Enabling all collection event reporting is useful when characterizing a GEM interface. Disabling all collection events is useful before enabling the reporting of desired collection events.</p>
S2F33	Host -> Equipment	<p>Define one or more reports.</p> <p>An empty list will delete all reports as well as the report links to collection events. Deleting all reports is a useful when resetting the subscriptions, or when connecting to a GEM interface for the first time to override default subscriptions.</p>
S2F35	Host -> Equipment	<p>Link one or more reports to a set of collection events.</p> <p>If reports are already linked to a collection event, you have to remove them and then link all collection events in one message. An empty list will remove report links from the collection event.</p>
S1F23	Host -> Equipment	<p>Request the list of available collection events and the available data for each collection event.</p>
S6F11	Equipment -> Host	<p>The collection event message.</p> <p>If no reports are linked, the message will only include the collection event ID number. If one or more reports are linked to the collection event, then the report data for each linked report will be included in the message.</p>



Frequently Asked Questions about Collection Events

1. How much bandwidth do collection events require?

This depends on several factors.

- a) The number of collection events that are enabled by the host.
- b) The size of the data reports linked to the collection events.
- c) The frequency at which the enabled collection events are triggered by the equipment. This depends on the meaning of the collection event.

2. How fast can collection events be triggered?

The GEM standard does not limit collection event frequency and uses standard communication hardware. In other words, by improving the hardware you can allow for faster collection events.

GEM allows for two protocols: SECS-I and HSMS. SECS-I is based on RS-232 serial communication and therefore little used today. Such implementations are not able to trigger collection events very quickly. HSMS is based on network communication. Because serial communication is slow, by far most GEM implementations use HSMS. GEM uses TCP/IP very efficiently. The possible frequency of collection events depends on the speed of the network hardware, equipment computer performance, and host computer performance. Like most protocols, it usually takes more computer resources to consume messages than it does to produce them.

The speed at which collection events can be generated also depends on the data reports linked to the collection events. For example, if a data report is large, like 10 MB, this will impact performance.

3. Why aren't I receiving the collection event messages?

There are a few reasons why a host might not receive collection event messages.

- a) Host and equipment must have established GEM communication using a successful S1F13/S1F14 exchange.
- b) GEM control state must be on-line. It cannot be in a host-offline or equipment-offline state.
- c) GEM spooling must be inactive. To disable spooling while it is active will not make spooling go inactive. If the spooled messages are not wanted, then purge spooling using message S6F23. If the spooled messages are wanted, then request them iteratively using S6F23 until the spooling state becomes inactive.
- d) The collection event must be enabled. Use S1F3 to check the "EventsEnabled" status variable to confirm that the collection event is enabled. Use message S2F37 to enable the collection event.
- e) The collection event activity needs to occur. For example, a collection event reporting when material arrives will never occur if material does not actually arrive. If the activity happens and the above conditions are satisfied, then the equipment's GEM interface has a defect.



Frequently Asked Questions about Collection Events (Continued)

4. What if an equipment's GEM interface does not publish the collection event I need?

Ask the equipment supplier to add the desired collection event. It is difficult for an equipment supplier to accurately predict all collection events that the factories will want. The equipment supplier will need to upgrade their GEM interface software at the factory.

5. How large of data reports can be when linked to a collection event?

GEM allows a single data variable value or status variable value to be an array or structure of any data type including a floating point, string or integer. A single array is limited to 16.777215 MB. Total message size is limited to 4.294967295 GB.

Conclusion

The GEM collection event feature is key to successfully implementing and using the GEM standard. It is the basis for tracking equipment activity and an extremely efficient and effective method for collecting activity related information from the equipment. If you are implementing a GEM interface, start by implementing collection events. If you are using a GEM interface, subscribe to its collection events and see how useful they are.

Chapter 3

Data Polling

GEM is an industry standard, which defines standard methods to communicate between process equipment and factory host software for monitoring and controlling purposes. By connecting GEM equipment, factories can immediately experience operational benefits. Factory hosts can collect data in multiple ways. A previous chapter discussed collecting data by using collection event reports where data is pushed to the host based state transitions performed by the equipment. In addition to event reports, the factory host often has a need to poll the equipment for current data values. Data values can be directly requested by the host or can be sampled on a periodic basis in a trace report. This is called Data Polling and is the topic for today's discussion.

Types of Data

There are three types of data in a GEM interface:

- **Data Variable (DV)** – data items that can be gathered when an equipment event occurs. This data is only guaranteed to be valid in the context of the event. For example, the GEM interface may provide an event called PPChanged (triggered when a recipe changes). The interface may also provide a data variable called changed recipe. The value of this DV is only valid in the context of the PPChanged event. Polling the value at a different time may have invalid or unexpected data.
- **Status Variable (SV)** – data items that contain information about the equipment. This data is guaranteed to be valid at any time. For example, the equipment may have a temperature sensor in a process module. The GEM interface may provide a ModuleTemperature status variable. The host can request the value of this SV at any time and expect the value to be accurate.
- **Equipment Constant (EC)** – data items that contain equipment settings. Equipment Constants determine how equipment will behave. For example, a GEM interface may have an equipment constant called MaxSimultaneousTraces which specifies the maximum number of traces that can be requested simultaneously from the host. The value of equipment constants is always guaranteed to be valid and up to date.



Data Properties

Each of the three data types listed above have similar properties that help define the data. The equipment supplier is responsible for providing these properties in a GEM manual so that the factory host will be able to interact with the data. Some of the important data properties are:

- ID – a numeric ID that must be unique in the GEM interface. These IDs can be grouped by data type and are referred to as SVIDs (Status Variable IDs), DVIDs (Data Variable IDs) and ECIDs (Collection Event IDs).
- Name – a human-readable name for the data item
- Format – the data type of the item.
 - Data formats can be simple (numeric, ASCII, Boolean) or complex (arrays, lists, structures). For example, numeric types can be I1, I2, I4, I8 (signed integer types of different byte length), U1, U2, U4, U8 (unsigned integer types) and F4 or F8 (floating point types).
 - List and array types contain multiple values in the data item. For example, image data would be formatted as a byte array.
 - Structure types contain a specific type of data. For example, a variable may represent a slot map which contains carrier information as well as a list of slots and their wafer placement status.
- Value – the actual value of the data item. Data values are in an accurate, efficient, self-describing binary format so that the host will know how to interpret the data. The data format allows for collection of more data more efficiently.

Data Polling

As mentioned, the factory host will often get data on a regular basis through trace reports and event reports that it defines. GEM also provides a method for the factory host to poll data based on its needs.

Status Variables

The host can query the value of status variables at any time by sending an S1F3 message containing a list of SVIDs for which to obtain the value. If the list has a length of one, only the value of the single SV will be returned. If the list has a length of zero, the values of all SVs defined in the interface will be returned. The values are returned in a list in an S1F4 message from the equipment.

The host can also request a list of SV names from the equipment by sending an S1F11 message to the equipment. The list rules mentioned above apply to this message as well. The return message will contain an entry for each SV that displays its SVID, Name and Unit.

Equipment Constants

An equipment's GEM interface can publish many different collection events. The host will not typically want to be notified of all of them at once and it does not have to. Collection events use a publish/subscribe design pattern in two ways.

Equipment Constants

Similar to the way SVs work, the host can also query the values of equipment constants defined in the GEM interface by sending an S2F13 message. The values will be returned from the equipment using an S2F14 message.

Also similar to SVs, the names of ECs can be queried using an S2F29 message.

Data Variables

Since data variables are only valid in the context of a collection event, there is not a method for polling values of data variables. The value of a Data Variable can only be reported in a collection event report.

Other

In addition to the methods for polling data discussed above, the following items can also be obtained from a GEM host by polling the equipment:

- Collection Events (CE) – The host can query what Collection Events are available on the GEM interface along with what DVs are associated with each CE. These are requested using the S1F23 message.
- Alarms – The host can query what Alarms are available on the system by sending an S5F5 message listing the ALIDs of the desired alarms. The return message lists the alarm code and alarm text associated with the ALID. Two status variables are required to be present in every GEM interface. AlarmsEnabled contains a list of IDs of all enabled alarms on the equipment. AlarmsSet contains the list of ID for alarms on the equipment that are currently in the Set state. Since these values are status variables, they can be queried at any time.
- MDLN and SOFTREV – The response to an S1F1 (Are you there?) message will contain the equipment model type (MDLN) and software revision (SOFTREV) for the equipment.
- DateTime – The date and time for the equipment can be requested using an S2F17 message. The host can synchronize the equipment's time using the S2F31 message. GEM requires the equipment to maintain a Clock SV containing the current time. Allowing the host to query and synchronize time provides the capability to order nearly simultaneous events on the system.



Trace Data Collection

Trace data collection provides a method of sampling data on a periodic basis. The time-based approach to data collection is useful in tracking trends or repeated applications within a time window or monitoring of continuous data.

When creating a trace definition, the host provides the following:

- Sample period – the time between samples. The resolution is in centiseconds, so it is possible to gather data very quickly using a trace. It is common for equipment to support as fast as a 10 Hz trace interval.
- Group size – number of samples included in a trace report
- SVIDs – List of status data to be included in the trace
- Trace request id – identifier of the trace request (GEM only allows trace IDs of type integer)

The host defines a trace request by using the S2F23 message. Trace reports are sent from the equipment to the host using the S6F1 message.

Trace Sample

Let's suppose that a piece of equipment is processing a wafer and the processing takes 5 minutes. It is important to keep the chuck temperature within a certain acceptable range and to make sure that the chamber pressure stays below a specified level. It is sufficient to monitor the values at 15 second intervals, but we can create groups of data to only receive reports once a minute. The host could send an S2F23 message with the following trace configuration:

Trace ID: 100 (ID must be an integer)

Sample Period: 00001500 (take a sample every 15 seconds)

Total Samples: 75 (Samples every 15 seconds for 5 minutes)

Group Size: 4

SVID List:

300 (ID of the status variable that contains information about chuck temperature)

301 (ID of the status variable that contains information about chamber pressure)

After one minute, the first trace report will be delivered using an S6F1 message from the equipment.

The message will contain the following information:

100 (Trace ID)

4 (last sample number)

2018-01-22T14:20:34.8 (date format depending on TimeFormat equipment constant)

Status Value List: (Length is 8: 2 SVs with a group size of 4)

219.96 (chuck temperature for first sample)

0.0112 (pressure for first sample)

219.97 (chuck temperature for second sample)

0.0122 (pressure for second sample)

219.97 (chuck temperature for third sample)

0.0120 (pressure for third sample)

219.96 (chuck temperature for fourth sample)

0.0119 (pressure for fourth sample)

After another minute the trace report may look like the following:

100 (Trace ID)

8 (last sample number)

2018-01-22T14:21:34.8 (date time shows one minute later than the first trace)

Status Value List: (Length is 8: 2 SVs with a group size of 4)

219.96 (chuck temperature for fifth sample)

0.0112 (pressure for fifth sample)

219.97

0.0122

220.01

0.0120

220.00

0.0119

Three more reports will be received at one-minute intervals. The host can check returned values in the report and react accordingly if values are out of the expected range.

Conclusion

The host could poll status data using S1F3 if it wanted to check a value at a given point in time. It can set up a trace if it wants to continuously collect data over a given period of time.

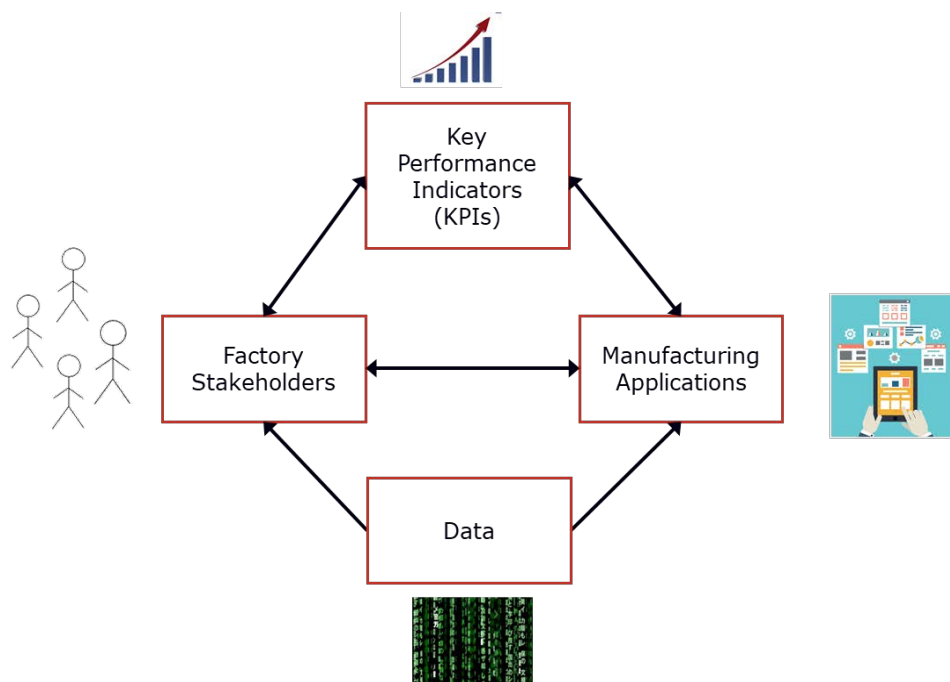
Using the data sampling methods outlined in this chapter will allow host applications to poll the data they need when they need it. GEM provides flexibility in requesting data from the equipment either by allowing the host to query values at a given point in time or to be sampled on a periodic basis using a trace.

Chapter 4

GEM Factory Application Support

Unlike the previous chapters which deal with specific features and capabilities of the SEMI E30 GEM (Generic Equipment Model) standard, this chapter identifies a number of the factory applications that depend on collecting data from the equipment.

Moreover, since we often hear the question “How do the factories actually use the different types of equipment information we’re expected to provide?” this chapter will summarize the specific data required to support a number of these applications. This list is by no means exhaustive, but should give you an idea of the range of factory stakeholders whose objectives are supported by GEM data collection. The figure below illustrates the relationship between the Key Performance Indicators (KPIs), the factory stakeholders responsible for optimizing them, the applications used to achieve this, and data required by these applications.



The most effective way to share this kind of information is in tabular form. Within a group of related applications (e.g., scheduling, preventive maintenance), the applications are listed in generally increasing order of complexity, which is also the likely order of implementation by the factory applications development staff.

What do factories DO with all that data?

Factory Application	Equipment Data Required
OEE (Overall Equipment Effectiveness)	Transition events and status codes sufficient to classify equipment states for all time periods
Intra-equipment material flow	Material tracking events; material location state indicators and state change events
Process execution tracking	Start/stop events for all processing modules; recipe step indicators and step change events for all processing modules that support multi-step recipes
WTW (wait time waste) analysis	The combination of events required for the intra-equipment material flow and process execution tracking applications (see above) and context data required to classify material states for all time periods (see the SEMI E168 Product Time Management standard for a deeper explanation)
Time-based PM (Preventive Maintenance)	Run timers at the FRU (field replaceable unit) level
Usage-based PM	Usage parameters and accumulators appropriate for each FRU, such as time-in-state, execution cycles, fluid flow rates, consumables flow rates, power consumption, etc.
Condition-based PM	Meaningful "health indicators" for each FRU
FDC (Fault Detection and Classification)	Equipment/process parameters required by specific fault models and associated context information (this is difficult to do completely because most FDC models are "trained" with knowledge of "good" and "bad" runs, which is not known to the equipment supplier a priori)
Automated equipment interdiction	Remote stop command (e.g., issued by an FDC application sensing an existing or imminent fault)
Equipment configuration monitoring	Vector of important equipment constants with expected values and acceptable ranges; may need to support multiple sets, if the values are setup-dependent. Designed to catch human errors resulting from operator manual adjustments
Component fingerprinting	Performance parameters for key equipment mechanisms, including command/response signals at the sensor/actuator level
Static job scheduling	Setup and execution times per product/recipe combination and current setup information
Real-time job dispatching	Estimate of current job completion time; estimate of completion time for all material queued at the equipment
Factory cycle time optimization	Material buffer contents, job queue information
Operator notification	Notification codes for frequent operator actions in a non-/semi-automated environment, such as load/unload material, select/confirm recipe, provide manual "assist" if the equipment is stuck, etc.
Real-time dashboard	Equipment/component production status indicators
Equipment failure analysis	Meaningful alarm/fault codes and perhaps recent history/statistics
Run-to-run process control	Identification of recipe adjustable parameters and commands to remotely update them

To the extent that some of the application data described in the table above can be standardized across equipment types, there is an opportunity to create generic factory applications that would only require a mapping from the supplier-specific GEM IDs (collection event IDs, status/data variables, equipment constants, etc.) to their generic counterparts.



Conclusion

We hope this explanation helps you appreciate how valuable equipment information is for the factories that consume it, and therefore how important it is to provide a rich set of events, variables, and other detailed information in the GEM interfaces you design in the future.

Chapter 3

Alarms

Previous chapters have talked about functionality that allows data to be collected through the GEM interface so the factory applications described chapter 4 can analyze this data. With this chapter, we return to a discussion of specific features and capabilities of the SEMI E30 GEM (Generic Equipment Model) standard, specifically the management of error conditions on the equipment.

In a perfect world everything goes according to plan, but in reality, things always go wrong. The secret to success is being able to know when something goes wrong, and then responding appropriately.

Just like a home alarm system, semiconductor fabs want to know when something bad has happened. They want to prevent the material being processed from being scrapped. Alarm management enables the equipment to notify the host when something goes wrong, and provide information about what has gone wrong. The GEM standard defines Alarm Management as the capability to provide host notification and management of alarm conditions occurring on the equipment.

In GEM, an alarm is any abnormal situation on the equipment that may endanger people, equipment, or material being processed. For example, if a technician opens an access panel to replace a component, the equipment should send an alarm notifying the host that it is not safe to operate the equipment in its current condition. Another example might be if an equipment requires a high temperature for processing but a sensor detects a low temperature condition, it should trigger an alarm, since running the process under those conditions could damage the material being processed.

It is also the responsibility of the equipment manufacturer to inhibit unsafe activities on the equipment when an alarm condition is present. The equipment manufacturer knows best what specific alarms are required on the equipment to ensure safety for people, equipment and material.



Often it is useful to have more information about the conditions in the equipment at the time an alarm condition occurs. Communicating that additional information to the host is valuable, but cannot be done through the normal Alarm Report Send/Acknowledge messages. To provide a way to get this additional information, GEM requires that two collection events be defined for each possible alarm condition on the equipment – one event for when the alarm is set, and another for when the alarm is cleared. These collection events allow the GEM event data collection mechanisms to be used to send the additional related information to the host when an alarm changes state.



In addition to providing the time of an alarm state change, Alarm Management on the equipment must allow the Host to request a list of all alarm IDs and associated alarm text. The host must also be able to enable/disable individual alarms on the equipment, and query the equipment for the list of alarms that are currently enabled for reporting.

The state diagram for an Alarm is not very exciting, but it fills a vital need. The image below illustrates the Alarm State diagram:



GEM alarms only have 2 states: each alarm is either SET or CLEAR. It's simple but effective.

Conclusion

Alarm Management isn't rocket science, but through effective use of Alarm Management, fabs can carefully monitor the health of their process equipment and minimize negative impacts to their production yield.

Chapter 6

Recipe Management

Following the last several chapters on GEM features and benefits, including Collection Events, Data Polling and Alarms, we now get into the features and merit of the GEM feature called Recipe Management. This chapter will cover the definition of recipes, what recipe management means and why you need this feature!

What are recipes?

Recipes are sets of instructions describing how the equipment should process its material. The Recipe content is defined by the equipment supplier.

What is recipe management?

Recipe management allows the factory host to transfer recipes to and from the equipment. It also requires the equipment to notify the factory host when recipes are changed on the equipment.

Why do you need this feature?

Almost all semiconductor factories require this feature to ensure recipe integrity and to support traceability.

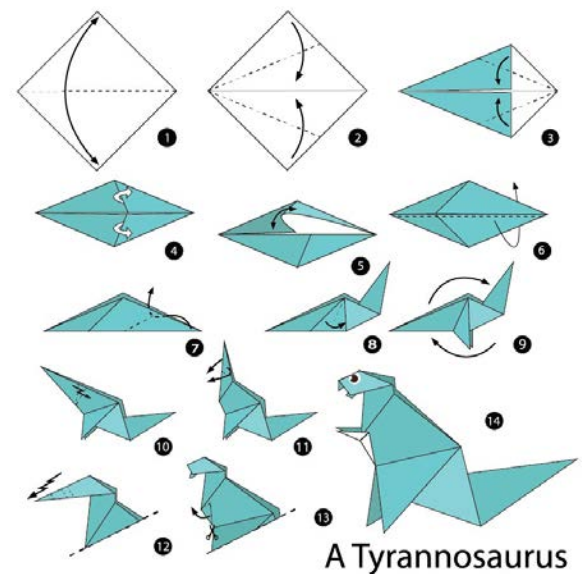
The host will upload approved recipes from the equipment and save them for later use to ensure that the recipe does not change. For traceability, the recipe is usually saved with the process data.

How does recipe management work?

Recipes are passed between the host and equipment via SECS messages. There are several sets of SECS messages to enable this. E30 GEM specifies formatted, unformatted, and large recipe message sets. The large recipe message set will not be discussed here.

The equipment is also required to notify the host whenever recipes are changed by an operator at the equipment. A PPChange collection event is generated with two data variables PPChangeName containing the PPID of the recipe that changed and PPChangeStatus containing the type of change (created, deleted, edited).

Once a recipe has been transferred to the equipment, the equipment should verify the content. If the recipe is invalid, then a PPVerificationFailed collection event should be generated with a PPErrror data variable containing the validation failure information to notify the host of the problem. The recipe should not be used if it fails verification.



Identification

Each recipe is identified by an ASCII name called a process program ID or PPID. The factory host and the equipment GEM interface use the name in recipe operations.

Persistence

Recipes are persisted in a GEM interface. If the host disconnects and reconnects, or if the equipment is restarted, the GEM interface will remember the recipes. In addition, most factory hosts will save recipes on the factory side.

Which messages are used?

Here is a summary of each of the primary messages related to collection events. Note that the "S" identifies the "stream" and "F" identifies the "function". Together, a stream and function number uniquely identify a message.

All Recipes

Message ID	Direction	Description
S7F17	Host -> Equipment	Delete a recipe from the equipment. An empty list will delete all recipes from the equipment.
S7F19	Host->Equipment	Request a list of available recipes from the equipment

Unformatted Recipes

Message ID	Direction	Description
S7F1	Host<-Equipment	Equipment requests to upload a recipe
S7F3	Host<-Equipment	Equipment uploads a recipe to the host
S7F5	Host<-Equipment	Equipment requests a recipe from the host
S7F1	Host->Equipment	Host requests to download a recipe
S7F3	Host->Equipment	Host downloads a recipe to the equipment
S7F5	Host->Equipment	Host requests a recipe from the equipment

Formatted Recipes

Message ID	Direction	Description
S7F1	Host<-Equipment	Equipment requests to upload a recipe
S7F23	Host<-Equipment	Equipment uploads a recipe to the host
S7F25	Host<-Equipment	Equipment requests a recipe from the host
S7F1	Host->Equipment	Host requests to download a recipe
S7F23	Host->Equipment	Host downloads a recipe to the equipment
S7F25	Host->Equipment	Host requests a recipe from the equipment
S7F29	Host<-Equipment	Equipment requests to verify a recipe
S7F27	Host<-Equipment	Equipment sends recipe verification results

How large a recipe can be transferred?

For unformatted recipe messages, the recipe is either a single ASCII string or a binary array value. A single array value is limited to 16.777215 MB.

Formatted recipe messages, the recipe is split up into a list of items. A single array value is limited to 16.777215 MB. Total message size is limited to 4.294967295 GB.

Conclusion

If an equipment has recipes, then it is imperative to implement GEM recipe management on that equipment's GEM interface even if the recipe content is just an "unformatted" recipe in XML format. Recipe management is a key to effective factory automation and the future of manufacturing for minimizing scrap caused by running the wrong recipe and for empowering the factories to minimize dependency on operator oversight.

Chapter 7

Documentation

As the introduction of this eBook on Features and Benefits of SECS/GEM points out, the SECS/GEM standards define a standardized interface that may be used on any equipment. A GEM interface exposes an equipment's capabilities through status variables, data variables, collection events, alarms, data formats, error codes, SECS-II messages, and other optional GEM capabilities. The GEM standard requires each equipment to come with documentation; ensuring a factory has the information it needs to use an equipment's GEM interface. This documentation is commonly referred to as the GEM manual.

The GEM manual may be distributed in many ways. Currently, most GEM manuals are provided digitally in a Word, Excel, or PDF document. The vast amount of information in a GEM manual is used to make purchasing decisions, develop host software, and test equipment. For a full GEM interface, the GEM manual must include the following topics: State Models, Scenarios, Data Collection, Alarm management, Remote Control, Equipment Constants, Process Recipe Management, Material Movement, Terminal Services, Error Messages, Clock, Spooling, Control, Supported SECS-II messages, GEM Compliance statement, and Data Item Formats. To keep this chapter a reasonable length, we will only cover a few of the required topics.



GEM Compliance Statement

The compliance statement is one of the first topics to be reviewed. It is a quick and easy way to understand the features of an equipment's interface. The manufacturer is required to mark which GEM capabilities are implemented on the equipment, and if they are implemented in a way that is compliant with the GEM standard.

Table 11 GEM Compliance Statement

GEM COMPLIANCE STATEMENT		
<i>FUNDAMENTAL GEM REQUIREMENTS</i>	<i>IMPLEMENTED</i>	<i>GEM-COMPLIANT</i>
State Models	<input type="checkbox"/> Yes <input type="checkbox"/> No	<input type="checkbox"/> Yes ^{#1} <input type="checkbox"/> No
Equipment Processing States	<input type="checkbox"/> Yes <input type="checkbox"/> No	
Host-Initiated S1,F13/F14 Scenario	<input type="checkbox"/> Yes <input type="checkbox"/> No	
Event Notification	<input type="checkbox"/> Yes <input type="checkbox"/> No	
On-Line Identification	<input type="checkbox"/> Yes <input type="checkbox"/> No	
<i>FUNDAMENTAL GEM REQUIREMENTS</i>	<i>IMPLEMENTED</i>	<i>GEM-COMPLIANT</i>
Error Messages	<input type="checkbox"/> Yes <input type="checkbox"/> No	<input type="checkbox"/> Yes ^{#1} <input type="checkbox"/> No
Documentation	<input type="checkbox"/> Yes <input type="checkbox"/> No	
Control (Operator Initiated)	<input type="checkbox"/> Yes <input type="checkbox"/> No	
<i>ADDITIONAL CAPABILITIES</i>	<i>IMPLEMENTED</i>	<i>GEM-COMPLIANT</i> ^{#2}
Establish Communications	<input type="checkbox"/> Yes <input type="checkbox"/> No	<input type="checkbox"/> Yes <input type="checkbox"/> No
Dynamic Event Report Configuration	<input type="checkbox"/> Yes <input type="checkbox"/> No	<input type="checkbox"/> Yes <input type="checkbox"/> No
Data Variable and Collection Event Namelist Requests	<input type="checkbox"/> Yes <input type="checkbox"/> No	<input type="checkbox"/> Yes <input type="checkbox"/> No
Variable Data Collection	<input type="checkbox"/> Yes <input type="checkbox"/> No	<input type="checkbox"/> Yes <input type="checkbox"/> No
Trace Data Collection	<input type="checkbox"/> Yes <input type="checkbox"/> No	<input type="checkbox"/> Yes <input type="checkbox"/> No
Status Data Collection	<input type="checkbox"/> Yes <input type="checkbox"/> No	<input type="checkbox"/> Yes <input type="checkbox"/> No
Alarm Management	<input type="checkbox"/> Yes <input type="checkbox"/> No	<input type="checkbox"/> Yes <input type="checkbox"/> No
Remote Control	<input type="checkbox"/> Yes <input type="checkbox"/> No	<input type="checkbox"/> Yes <input type="checkbox"/> No
Equipment Constants	<input type="checkbox"/> Yes <input type="checkbox"/> No	<input type="checkbox"/> Yes <input type="checkbox"/> No
Process Recipe Management	<input type="checkbox"/> Yes <input type="checkbox"/> No	Process Programs: <input type="checkbox"/> Yes <input type="checkbox"/> No E42 Recipes: <input type="checkbox"/> Yes <input type="checkbox"/> No E139 Recipes: <input type="checkbox"/> Yes <input type="checkbox"/> No
Material Movement	<input type="checkbox"/> Yes <input type="checkbox"/> No	<input type="checkbox"/> Yes <input type="checkbox"/> No
Equipment Terminal Services	<input type="checkbox"/> Yes <input type="checkbox"/> No	<input type="checkbox"/> Yes <input type="checkbox"/> No
Clock	<input type="checkbox"/> Yes <input type="checkbox"/> No	<input type="checkbox"/> Yes <input type="checkbox"/> No
Limits Monitoring	<input type="checkbox"/> Yes <input type="checkbox"/> No	<input type="checkbox"/> Yes <input type="checkbox"/> No
Spooling	<input type="checkbox"/> Yes <input type="checkbox"/> No	<input type="checkbox"/> Yes <input type="checkbox"/> No
Control (Host-Initiated)	<input type="checkbox"/> Yes <input type="checkbox"/> No	<input type="checkbox"/> Yes <input type="checkbox"/> No

#1 Do not mark YES unless all fundamental GEM requirements are implemented and GEM-compliant.

#2 Additional capabilities may not be marked GEM-compliant unless the fundamental GEM requirements are GEM-compliant.

State Models

State Models is a fundamental GEM capability, and is therefore implemented on every equipment. This capability defines the Communication, Control, and spooling behavior of the equipment. A processing state model must be provided. However, it is not possible to define a processing state machine that can be used on every equipment. The processing behaviors that should be the same for all equipment are specified by the standard. Each state model must be documented with a state model diagram, a transition table, and a text description of every state. The consistent and detailed information about each state model enables a factory to start writing a host application as soon as they have the GEM manual.

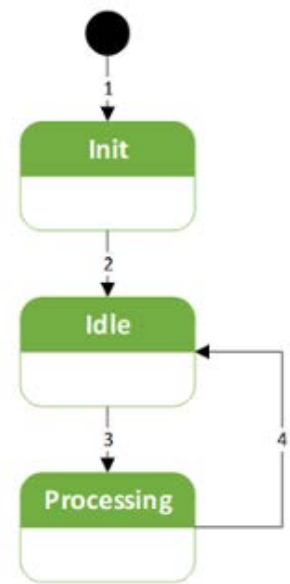


Figure 1, Example Processing State Model

Alarms, Collection Events, Equipment Constants, Data Variables, and Status Variables

Alarms, Collection Events, and Variables are large components in gathering data from an equipment. It should not be a surprise that these are required to be in the GEM manual. Each alarm on the equipment should have its ID, name, description, and associated Set/Clear events in the GEM manual. The documentation for each collection event should include the ID, name, description, and a list of associated variables. The documentation for all variables will include an ID, name, description, and the data type. Information about a variables default value or value range should also be provided when appropriate. Although not required, it is common to display all this information in five tables that are easy to find. There would be a single table for each of the following: alarms, collection events, equipment constants, data variables, and status variables. See the examples below.

Alarms

Name	ID	Set CEID	Clear CEID	Text
MyAlarm1	1	1001	2001	Alarm 1 Text

Collection Events

Name	ID	Description	Associated Data Variables
OperatorCommandIssued	6	Machine operator issued a control command.	OperatorCommand(6)
PPChange	7	A process program (recipe) has been created changed or deleted.	PPChangeName(3), PPChangeStatus(4)

Status Variables

Name	ID	Type	Unit	Min	Max	Default	Description
CONTROLSTATE	2028	U1		U1 0	U1 5	U1 4	State of the Control State Machine. Possible values include 1=EquipOffline 2=AttemptOnline 3=HostOffline 4=OnlineLocal and 5=OnlineRemote.

Remote Control

Once a factory can gather data from an equipment they start looking at how to control the equipment. Remote Control is the GEM capability that allows a host application to request an equipment to perform an action. Each remote command should be in the manual with its name, description, and details about each command parameter that may be sent with the command. The details of a command parameter should include the name, the format, and a description. An example is shown below.

Remote Control

Once a factory can gather data from an equipment they start looking at how to control the equipment. Remote Control is the GEM capability that allows a host application to request an equipment to perform an action. Each remote command should be in the manual with its name, description, and details about each command parameter that may be sent with the command. The details of a command parameter should include the name, the format, and a description. An example is shown below.

Name	Arguments		Processing States	Description
	Name (ASCII)	Value (ASCII)		
ABORT	AbortLevel	1	READY PROCESSING	<p>Command to terminate the current cycle prior to its completion. Abort intends to quickly stopping during abnormal conditions. Abort makes no guarantee about the subsequent condition of material. In the above example, the wafers being processed at the time of the abort may not be completely processed. The termination occurs at the next "safe break point," retrieves all material, stops in a safe condition and returns to the idle state in the processing state machine.</p> <p>The "AbortLevel" argument is optional. If omitted, the ABORT command responds the same as if an AbortLevel "1" argument was provided.</p>

SMN and SEDD

GEM manuals rarely come in a format that is easy to parse in software. This often results in duplicating code and making small changes in order to communicate with other equipment. SEMI E172 SECS Equipment Data Dictionary (SEDD) and E173 SECS Message Notation (SMN) are two standards that can drastically increase the flexibility and reusability of a host application. SEDD is an xml file that is easily distributed and parsed in software. SEDD can be considered a modernized GEM manual because it contains much of the same information that is found in a GEM manual. For example, a SEDD file contains details about every variable, collection event, alarm, and supported SECS-II message. A SEDD file uses SMN to represent the data items, variables, and SECS-II messages. SMN is also XML and is the first standard to define a notation for representing data items and SECS-II messages. This means a single application can read a SEDD file, have a short configuration process, and then immediately start using the GEM interface of an equipment. These features allow a single application to be used with multiple equipment instead of creating slightly different variants for each equipment.

Conclusion

The GEM manual is a crucial piece of documentation that is required by the GEM standard to be provided with every equipment. The GEM Manual should be the first place to look for an answer when there is a question about an equipment's GEM interface. SEMI continues to improve the content and flexibility of a GEM manual by updating existing standards and creating new standards.



Chapter 8

Equipment Terminal Services

After several chapters discussing data collection, events, alarms, recipe management and documentation, this Chapter focuses on the “Twitter” of the GEM standard – Equipment Terminal Services. We will examine what terminal services are, why they are needed and the mechanics of how they work.

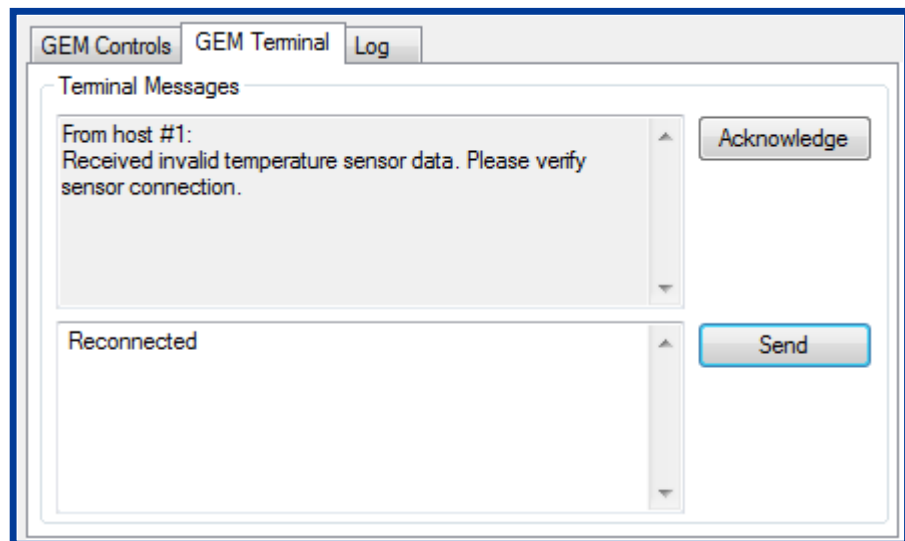
What are Terminal Services?

Equipment Terminal Services allows the factory operators to exchange information with the host from their equipment workstations. The host can display information on the equipment’s display device. It also allows the operator of the equipment to send information to the host. The equipment must be capable of displaying information passed to it by the host for the operator’s attention.

Why Do You Need This Feature?

An example of when terminal services might be used is as follows:

1. The host gets notified by the FDC software that the process module had an excursion that needs to be addressed.
2. The host turns on an operator notification light on the light tower. The notification light needs to be accompanied by a reason that the light was illuminated.
3. The host sends a terminal message saying that the FDC software detected an excursion and that the operator should address the issue.
4. Along with the signal tower light, the terminal services notification is active on the tool.
5. The operator sees and acknowledges the message.
6. Optional: There are different ways to recover, but the operator could send a terminal message to the host after the issue is resolved.



How Does Terminal Services Functionality Work?

When the host sends a terminal message to the equipment, the equipment is required to display the message to the operator. The display must be able to show up to 160 characters (even more than can be sent in a single tweet using Twitter) but may display more than that. The equipment's display device must have a mechanism for notifying the operator that a message was received and not yet recognized by the operator. The message continues to be displayed until the operator recognizes the message. The equipment must provide a method, such as a push button, for the operator to acknowledge the message. Message recognition by the operator results in a collection event that informs the host that the operator has received the information. The equipment application is not required to interpret the data sent from the host. It is solely information meant for the operator.

If the host sends a new message is sent before the operator acknowledges a previous message, the new message overwrites the previous message.

The host may clear unrecognized messages (including the indicator) by sending a zero-length message. The zero-length message is not considered an unrecognized message.

The equipment must also allow the operator to send information entered from the operator's equipment console to the host.

Which messages are used?

Message ID	Direction	Description
S10F3	H->E	Host sends textual information to equipment for display to the operator on a terminal
S10F1	H<-E	Operator sends text message to host
S10F5	H->E	(Optional) Host sends multi-block display message. If multi-block is not supported, the equipment responds with an S10F7 message that multi-block is not allowed.
S6F11	H<-E	Equipment sends collection event to the host notifying the host that it has recognized the message

Conclusion

Some equipment might not need terminal services, but it can be useful in some factories and in some situations. For example, it is particularly useful when the factory can detect specific errors, give details to the operator using terminal service messages, and ask the operator to fix them. Other useful cases exist where the terminal service messages are very useful. Terminal services is a great example of how complete the GEM interface is, and how useful in real factory situations.

Chapter 9

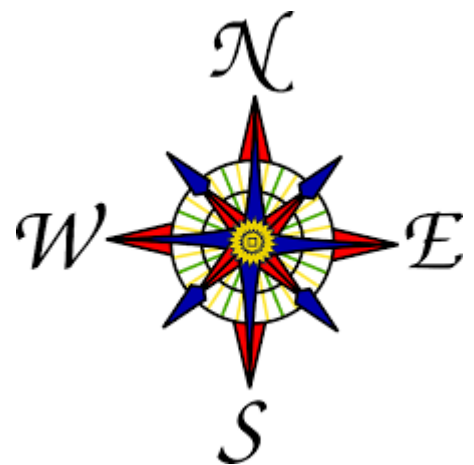
User Interface

Pretend you are a young hiker, planning a trip into a primitive area in the mountains. One of the first things you would learn about reading a map would be where to find the legend. The map legend contains important information needed to read a map, like indicating which direction is north. Once you know where to find the legend, you could orient the map so it made sense as you planned your hike.

Most equipment in a typical semiconductor or electronics assembly factory has a user interface that contains a lot of information about the equipment. Most equipment also contains many screens that are used for controlling or operating the equipment. With the use of GEM, a factory host system can control the equipment and collect important data generated during processing.

Like a map, there is a lot of information available on the user interface of a piece of equipment. It can sometimes be difficult to know where to find the important information the host system needs to properly control and communicate with the equipment. The GEM standards provide guidelines on how critical items on the equipment user interface should be presented and controlled. For example, if the host sends information to the equipment operator about tasks they need to perform, the GEM terminal message guidelines state that the information must remain on the user interface of the equipment until the operator acknowledges that they have read it.

The SEMI E30 standard defines the Specification for the Generic Model for Communications and Control of Manufacturing Equipment (GEM). In addition to providing the definition of the common set of equipment behavior and communication capabilities required for manufacturing automation, the standard also provides requirements on which items must be present on an equipment user interface and how they should be represented. User interface requirements spelled out by the standard address communication state, terminal service new message indicator, terminal services message recognition button, communications state default and communications state selector.



Conclusion

This may seem like a small thing, but just like knowing where to find the legend on a map enabled understanding of the lines and symbols on the map, so too the GEM standards can help provide an understanding of information presented on an equipment interface that is essential for communication with a factory host system.

Chapter 10

GEM Message Spooling Capabilities

Even the most robust computer networks experience communication failure. Regardless of the cause, a small outage could be responsible for a significant amount of mission critical data loss. GEM mediates this loss of data by providing the message spooling capability.

Spooling Definition

"Spooling is a capability whereby the equipment can queue messages intended for the host during times of communication failure and subsequently deliver these messages when communication is restored" SEMI E30-0717 7.12.

Spooling Benefits

Automated factories are data-driven. Data is extracted and analyzed to make decisions that influence how engineering and management teams react to ensure product yield is high and scrap is low.

Gaps in this data could lead to erroneous judgement or even guessing. Spooling is a backup system that ensures this data will be preserved and restored reducing the risk of losing valuable data.

GEM Capability Requirements

Spooling is not a GEM requirement however, if this additional capability is implemented it must be done so properly. Here are a few requirements for implementing a compliant spooling interface.

The equipment must provide the host with the ability to enable and disable spooling via the equipment constant "EnableSpooling". This EC is published by the equipment and the host can select the desired state.

When Spooling is implemented, it must be functional for all relevant primary messages and accessible using an S2, F43/F44 transaction. This excludes stream 1 messages which must be rejected if they attempt to "set spool".



Non-Volatile Storage

The equipment is responsible for allocating enough non-volatile-storage to store all messages that have been spooled for at least one processing cycle of the equipment. The NVS will also house all spooling-related status variables. NVS is used for this data so that if a power outage occurs the data is persisted.

Loss of Power

All messages that were spooled prior to the equipment's power loss will be available since they are persisted in non-volatile storage. All spooling context is restored from NVS if spooling was active at the time of the power loss occurred. This includes the spooled data as well as all spooling related status variables persisted in NVS.

Host responsibility for implementation of Spooling

Message spooling requires hosts to participate to successfully recover after a loss of communication. It is Ideal to leave spooling in the disabled state until the host has been programmed to properly handle all conditions that may occur in the entirety of this state machine. Disabled spooling is better than improperly managed spooling.

Once communication is re-established, the host must manage requesting the spooled messages. The host also has the option of purging the files from the equipment when necessary.

Conclusion

Though spooling is not a fundamental GEM requirement, if implemented it must be done so properly. Both host and equipment software have a responsibility to ensure GEM compliance when spooling is enabled. GEM spooling protects the potential loss of valuable data and provides a standard for both equipment and host software to adhere to with ease.

Chapter 11

Protocol Layer

T

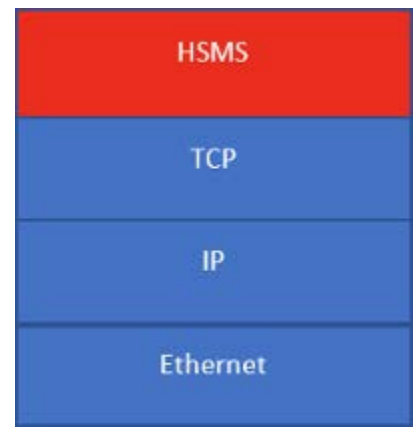
he protocol layer packages data and reliably transfers it between the factory host and the equipment GEM interface.

Protocol Layer Definition

The protocol layer implements the transport technology and data packing algorithms used to send messages across a wire between a factory host and an equipment GEM interface.

The SEMI E5 standard, SEMI Equipment Communications Standard 2 Message Content (SECS-II), defines SECS messages that are used as the data and defines how they are packed into binary buffers for transport.

The SEMI E37 and E37.1 standards, High-Speed SECS Message Services (HSMS), define a protocol for exchanging SECS messages over a TCP/IP connection. This is the most used transport technology in SECS/GEM.



HSMS Protocol Stack

The SEMI E4 standard, SEMI Equipment Communications Standard 1 Message transfer (SECS-I), defines a mechanism for exchanging SECS messages over RS-232. This is normally used for older equipment or for some hardware inside an equipment such as an EFEM controller.

The rest of the document will focus on SECS messages over HSMS.

Protocol Layer Benefits

The protocol layer in GEM maintains the connection and detects a loss of connection so either party may take appropriate action such as activating Spooling.

The protocol layer defines handshaking mechanisms to ensure delivery of messages if desired. The protocol layer connection is point-to-point between the factory host and equipment. It is a dedicated connection with no broadcast capabilities. This makes it easier to predict network loading.

Data Density

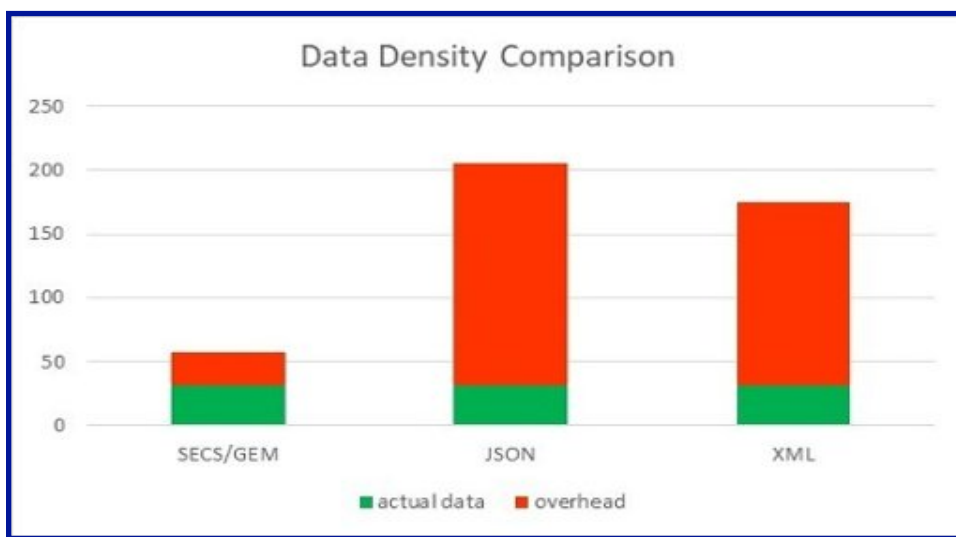
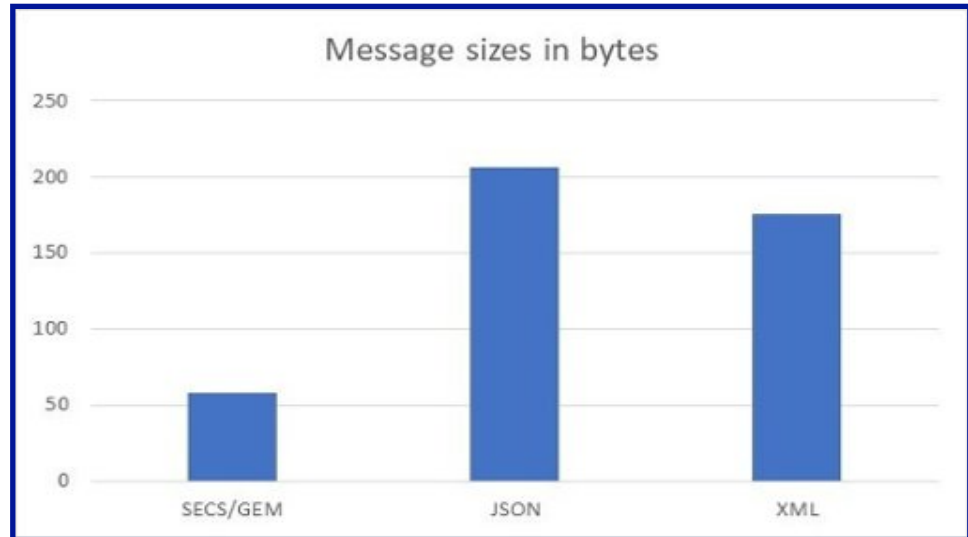
SECS/GEM transmits data with little overhead and high density. This means less network bandwidth usage for a given data set.

For illustrative purposes, let us look at a typical example of an event report and compare SECS/GEM messaging to a somewhat equivalent XML and JSON message.

Take a typical GEM interface that uses unsigned 4-byte integers for IDs and an event report containing 8-byte floating point numbers and 4-byte integers. An example of this message is shown in the table below in a SECS/GEM format per E5 and in equivalent JSON and XML formats.

SECS/GEM (readable E5 notation)	SECS/GEM wire format (binary bytes as hex)	JSON	XML
L U4 1 U4 2 L L U4 1 L F8 2.225073858507200 9E+8 F8 2.22*10-9 U4 79785324 U4 4	01 03 b1 04 00 00 00 01 b1 04 00 00 00 02 01 01 01 02 b1 04 00 00 00 01 01 04 81 08 41 aa 86 62 f3 b3 91 95 81 08 40 2a 66 66 66 66 66 68 b1 04 04 c1 6d 6c b1 04 00 00 00 04	{ Event: 2, Reports: [{ id: 1, data: [{ type: F8, value: 2.2250738585072009 E+8 }, { type: F8, value: 2.22*10-9 }, { type: U4, value: 79785324 }, { type: U4, value: 4 }] }] }	<Event id="2"> <Reports> <Report id="1"> <Data> <F8>2.225073 8585072009E+8 </F8> <F8>2.22*10- 9</F8> <U4>7978532 4</U4> <U4>4</U4> </Data> </Report> </Reports> </Event>

The binary SECS/GEM message will take 58 bytes over the wire, the JSON around 206 bytes and XML 175. The JSON and XML numbers can change a bit based on key/element names and the above is just one of many possible representations.

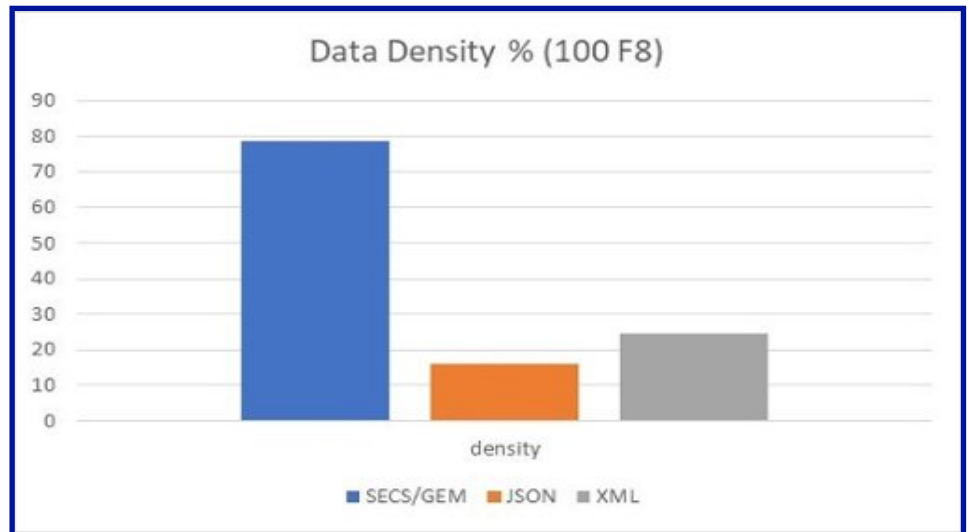


A chart showing the data density comparison for the example message is shown below. The Actual Data size is 2 4-byte integers + 2 8-byte floating point numbers + 1 4-byte event id + 1 4-byte report id = 32 bytes of actual data. The overhead is calculated by subtracting the actual data size from the total number of bytes for the message.

For the example message the data density for SECS the data density percentages are shown in the graph below. Data density percentage is calculated by the $(\text{actual data}) / \text{overhead} * 100$.



Now if we change the example message to have 100 8-byte floating point numbers in it, the Data Density % graph changes to the chart below. Notice the JSON and XML are relatively the same, but the SECS/GEM data density increases to 78%.



SECS/GEM encoding has very little overhead. The overhead for a message is 10 bytes for a header describing the message,

plus 1 to 4 bytes for the size of the message body. For any 4-byte integer or floating-point number in a SECS message, 6 bytes will be sent across the wire, 4 bytes for the integer value + 1 for the type + 1 for the length in bytes of the data. Likewise, for any 8-byte integer or floating-point number, 10 bytes will be sent. For a string value, the length will be the number of characters plus 2 to 4 bytes. Any time a List (L in the readable example above) appears in a SECS message, 2 to 4 bytes will be added to the message.

Arrays of numbers are brutally efficient in SECS/GEM. The overhead for an array is 1 byte for the type plus 1 to 4 bytes for the length of the array, plus the data in its native size. For example: an array of 10 4-byte integers would take 42 bytes, that is a data density of 95%!

In the JSON example, a 4-byte integer requires 16 bytes + the number of characters needed to represent the integer, so 17 to 28 bytes. Floating point numbers are the same overhead, but probably requiring more characters to represent the value.

In XML, the overhead is based on the sizes of the XML element names. Using the element names in the example above, for any 4 -byte integer the number of bytes across the wire will be 9 + number of characters needed to represent the integer, so 10 to 21 bytes. Floating point numbers are at the mercy of the string formatting used to represent the values.

In summary, looking at the per-item byte size across the wire, SECS/GEM is very dense. Take the 4-byte integer example where SECS/GEM is 6 bytes across the wire, the JSON example is 17 to 28 and the XML example is 10 to 21 bytes and you see as you scale the number of parameters the overhead really matters. 300mm Semiconductor equipment are expected to transfer 1000 parameters per second per process module to the host. For a 2-module equipment, this results in the following number of bytes just for the data: 12K bytes/ over SECS/GEM, 34K-56K for JSON, and 20K-42K for the XML example. These numbers do not account for size of the rest of the message, just the actual parts related to parameter values. If that data is transferred in lots of messages with few values per message, then the network load is even worse. Fewer, larger messages are always better in all cases.

XML and JSON may also add to the overhead depending on the transmission protocol used. For example, XML is often transmitted over HTTP using SOAP, this adds two additional layers of overhead and more bytes going across the wire for each message. The numbers of bytes shown for SECS/GEM are what is transmitted across the wire on top of TCP/IP.

No Data Translation

Numeric data is transmitted with no translation in SECS/GEM. Numbers are transmitted in their native format. For example: an 8-byte floating point number is transmitted in its 8-byte representation without any conversion, truncation, or rounding.

Any protocol such as JSON or XML must convert those 8-byte floating point numbers into a text representation. This takes computing resources for the encoding and decoding and significantly more bytes across the wire. IEEE754 requires 17 decimal digits to accurately represent an 8-byte floating point number as a string. Adding in characters for sign, decimal point, exponent and sign leads to 21 characters. That is over twice what SECS/GEM sends across the wire.

Circuit Assurance

HSMS defines a circuit assurance mechanism called Link Test. The protocol layer has a timer that is started if there are no active message exchanges. Every time the timer expires, a protocol message is exchanged to ensure the connection is still open.

Security

HSMS defines no security. There is no validation of the connecting party, no credentials or certificate is required to connect. The data is not encrypted by any normal encryption algorithms; however, data is obfuscated through the data packaging process and is not generally human readable.

Security is not normally seen as an issue since factory networks are isolated from the outside world.

Conclusion

The SECS/GEM Protocol Layer using HSMS provides a very efficient means of exchanging accurate data between the factory host and equipment.

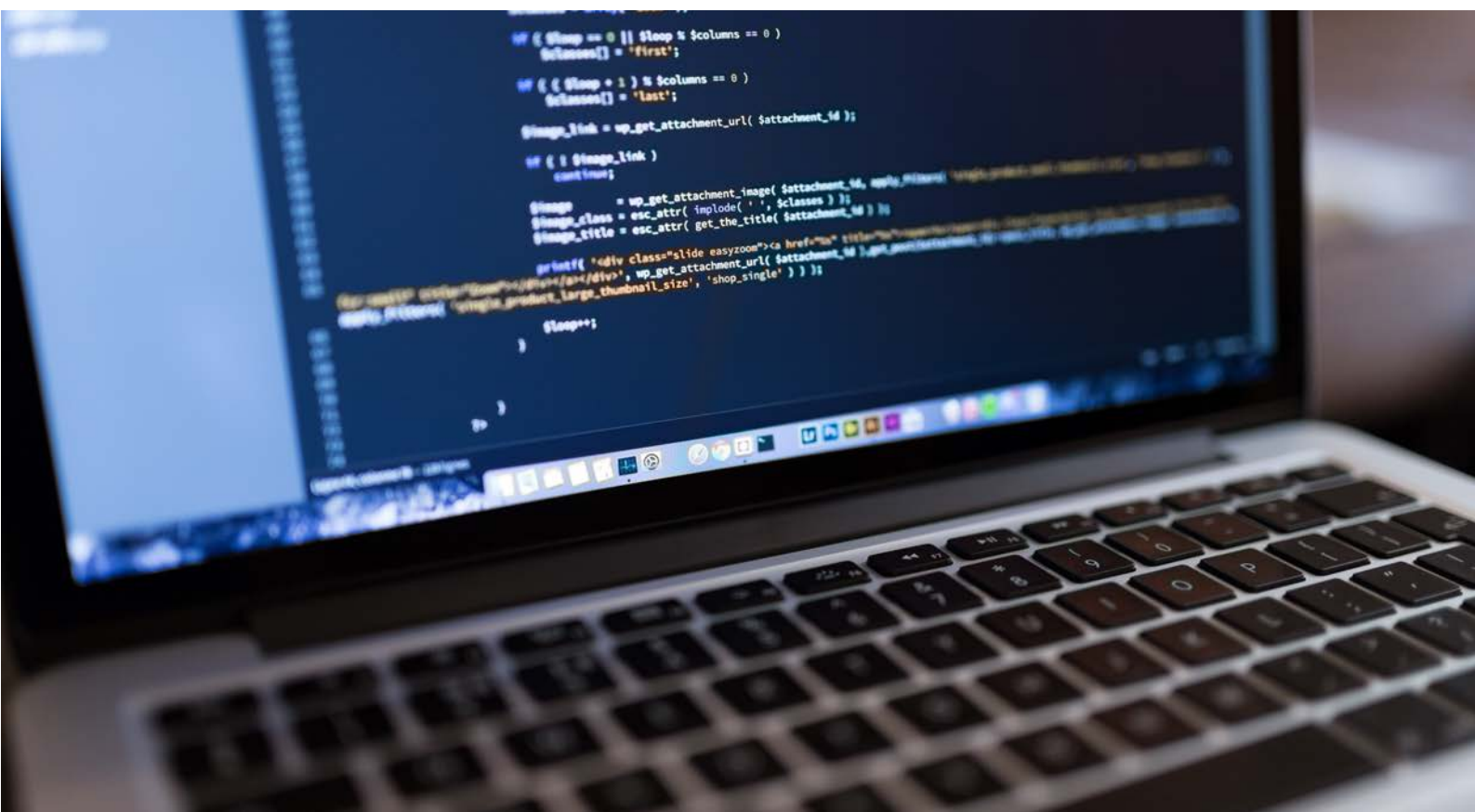
Chapter 12

Message Logging

Just like software logging is important for troubleshooting an application, logging the detailed message traffic between a factory host and the manufacturing equipment is just as important for troubleshooting.

For example, a host sends a command, and the equipment behaves based upon the message, but something does not work as expected. It would be very helpful to see the message that was sent and the reply from the equipment, in conjunction with any other logs from the equipment to determine where the problem is located.

The format used to display/represent the logged messages is also very important. The latest industry standard for SECS message formatting is SEMI E173, the Specification for XML SECS-II Message Notation (SMN).



Here is an example:

```
<?xml version="1.0" encoding="utf-8"?>
<SECSMessageScenario xmlns="urn:semi-org:xsd.SMN">
  <Comment time="2018-02-05T18:19:20.365Z">State Change
NotConnected</Comment>
  <Comment time="2018-02-05T18:19:20.400Z">State Change NotSelected</Comment>
  <HSMSMessage time="2018-02-05T18:19:20.394Z" sType="Select.req" direction="H to
E" txid="1">
    <Header>FFFF0000000100000001</Header>
  </HSMSMessage>
  <HSMSMessage time="2018-02-05T18:19:20.417Z" sType="Select.rsp" direction="E to
H" txid="1">
    <Header>FFFF0000000200000001</Header>
    <Description>Communication Established</Description>
  </HSMSMessage>
```

Here is an S5,F5 example:

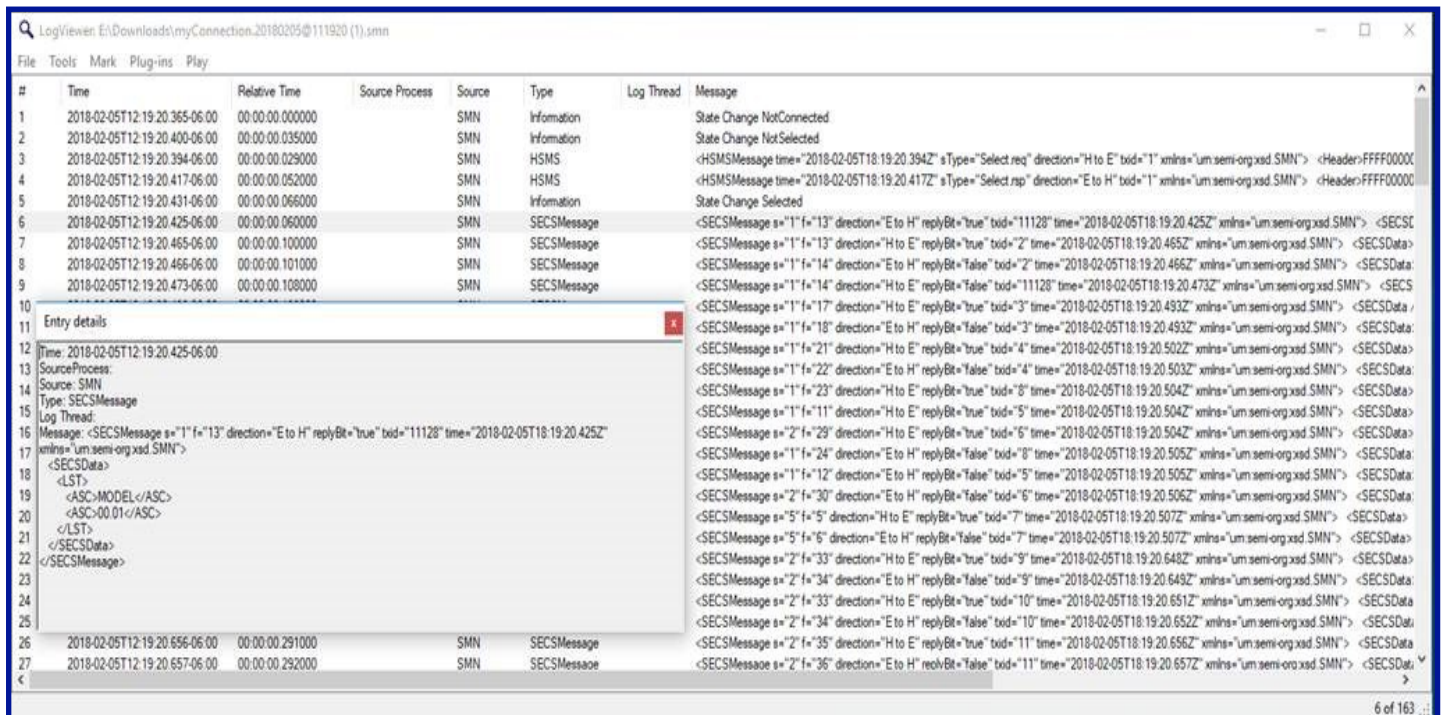
```
<SECSMessage s="5" f="5" direction="H to E" replyBit="true" txid="7" time="2018-02-
05T18:19:20.507Z">
  <SECSData>
    <UI4 />
  </SECSData>
</SECSMessage>
<SECSMessage s="5" f="6" direction="E to H" replyBit="false" txid="7" time="2018-02-
05T18:19:20.507Z">
  <SECSData>
    <LST>
      <LST>
        <BIN>0</BIN>
        <UI4>1</UI4>
        <ASC>Alarm 1 Text</ASC>
      </LST>
    </LST>
  </SECSData>
</SECSMessage>
```


The SMN format is ideally suited for:

- Capturing the HSMS header information in a clear way
- Logging messages in an exact, binary format
- Reading the logs using software
- Creating a host or equipment emulator, since it is easy to read the logging from a software application and play it back.
- Extracting data from the SMN logs

The logs can be captured by the Equipment, Host, or even a "network sniffer" like Cimetrix's CIMSniiffer utility.

Cimetrix's Logviewer utility supports SMN logs as well:



#	Time	Relative Time	Source Process	Source	Type	Log Thread	Message
1	2018-02-05T12:19:20.365-06:00	00:00:00.000000		SMN	Information		State Change NotConnected
2	2018-02-05T12:19:20.400-06:00	00:00:00.035000		SMN	Information		State Change NotSelected
3	2018-02-05T12:19:20.394-06:00	00:00:00.029000		SMN	HSMS		<HSMSMessage time="2018-02-05T18:19:20.394Z" sType="Select req" direction="H to E" bid="1" xmlns="um:semi-org.xsd:SMN"> <Header>FFFF0000
4	2018-02-05T12:19:20.417-06:00	00:00:00.052000		SMN	HSMS		<HSMSMessage time="2018-02-05T18:19:20.417Z" sType="Select rsp" direction="E to H" bid="1" xmlns="um:semi-org.xsd:SMN"> <Header>FFFF0000
5	2018-02-05T12:19:20.431-06:00	00:00:00.066000		SMN	Information		State Change Selected
6	2018-02-05T12:19:20.425-06:00	00:00:00.060000		SMN	SECSMessage		<SECSMessage s="1" f="13" direction="E to H" replyBt="true" bid="11128" time="2018-02-05T18:19:20.425Z" xmlns="um:semi-org.xsd:SMN"> <SECS
7	2018-02-05T12:19:20.465-06:00	00:00:00.100000		SMN	SECSMessage		<SECSMessage s="1" f="13" direction="H to E" replyBt="true" bid="2" time="2018-02-05T18:19:20.465Z" xmlns="um:semi-org.xsd:SMN"> <SECSData
8	2018-02-05T12:19:20.466-06:00	00:00:00.101000		SMN	SECSMessage		<SECSMessage s="1" f="14" direction="E to H" replyBt="false" bid="2" time="2018-02-05T18:19:20.466Z" xmlns="um:semi-org.xsd:SMN"> <SECSData
9	2018-02-05T12:19:20.473-06:00	00:00:00.108000		SMN	SECSMessage		<SECSMessage s="1" f="14" direction="H to E" replyBt="false" bid="11128" time="2018-02-05T18:19:20.473Z" xmlns="um:semi-org.xsd:SMN"> <SECS
10							<SECSMessage s="1" f="17" direction="H to E" replyBt="true" bid="3" time="2018-02-05T18:19:20.493Z" xmlns="um:semi-org.xsd:SMN"> <SECSData
11							<SECSMessage s="1" f="18" direction="E to H" replyBt="false" bid="3" time="2018-02-05T18:19:20.493Z" xmlns="um:semi-org.xsd:SMN"> <SECSData
12							<SECSMessage s="1" f="21" direction="H to E" replyBt="true" bid="4" time="2018-02-05T18:19:20.502Z" xmlns="um:semi-org.xsd:SMN"> <SECSData
13							<SECSMessage s="1" f="22" direction="E to H" replyBt="false" bid="4" time="2018-02-05T18:19:20.503Z" xmlns="um:semi-org.xsd:SMN"> <SECSData
14							<SECSMessage s="1" f="23" direction="H to E" replyBt="true" bid="8" time="2018-02-05T18:19:20.504Z" xmlns="um:semi-org.xsd:SMN"> <SECSData
15							<SECSMessage s="1" f="11" direction="H to E" replyBt="true" bid="5" time="2018-02-05T18:19:20.504Z" xmlns="um:semi-org.xsd:SMN"> <SECSData
16							<SECSMessage s="2" f="29" direction="H to E" replyBt="true" bid="6" time="2018-02-05T18:19:20.504Z" xmlns="um:semi-org.xsd:SMN"> <SECSData
17							<SECSMessage s="1" f="24" direction="E to H" replyBt="false" bid="8" time="2018-02-05T18:19:20.505Z" xmlns="um:semi-org.xsd:SMN"> <SECSData
18							<SECSMessage s="1" f="12" direction="E to H" replyBt="false" bid="5" time="2018-02-05T18:19:20.505Z" xmlns="um:semi-org.xsd:SMN"> <SECSData
19							<SECSMessage s="2" f="30" direction="E to H" replyBt="false" bid="6" time="2018-02-05T18:19:20.506Z" xmlns="um:semi-org.xsd:SMN"> <SECSData
20							<SECSMessage s="5" f="5" direction="H to E" replyBt="true" bid="7" time="2018-02-05T18:19:20.507Z" xmlns="um:semi-org.xsd:SMN"> <SECSData
21							<SECSMessage s="5" f="6" direction="E to H" replyBt="false" bid="7" time="2018-02-05T18:19:20.507Z" xmlns="um:semi-org.xsd:SMN"> <SECSData
22							<SECSMessage s="2" f="33" direction="H to E" replyBt="true" bid="9" time="2018-02-05T18:19:20.648Z" xmlns="um:semi-org.xsd:SMN"> <SECSData
23							<SECSMessage s="2" f="34" direction="E to H" replyBt="false" bid="9" time="2018-02-05T18:19:20.649Z" xmlns="um:semi-org.xsd:SMN"> <SECSData
24							<SECSMessage s="2" f="33" direction="H to E" replyBt="true" bid="10" time="2018-02-05T18:19:20.651Z" xmlns="um:semi-org.xsd:SMN"> <SECSData
25							<SECSMessage s="2" f="34" direction="E to H" replyBt="false" bid="10" time="2018-02-05T18:19:20.652Z" xmlns="um:semi-org.xsd:SMN"> <SECSData
26	2018-02-05T12:19:20.656-06:00	00:00:00.291000		SMN	SECSMessage		<SECSMessage s="2" f="35" direction="H to E" replyBt="true" bid="11" time="2018-02-05T18:19:20.656Z" xmlns="um:semi-org.xsd:SMN"> <SECSData
27	2018-02-05T12:19:20.657-06:00	00:00:00.292000		SMN	SECSMessage		<SECSMessage s="2" f="36" direction="E to H" replyBt="false" bid="11" time="2018-02-05T18:19:20.657Z" xmlns="um:semi-org.xsd:SMN"> <SECSData

Entry details

Time: 2018-02-05T12:19:20.425-06:00

Source Process: SMN

Source: SMN

Type: SECSMessage

Log Thread:

Message: <SECSMessage s="1" f="13" direction="E to H" replyBt="true" bid="11128" time="2018-02-05T18:19:20.425Z" xmlns="um:semi-org.xsd:SMN">

<SECSData>

<LST>

<ASC:MODEL</ASC>

<ASC:00 01</ASC>

</LST>

</SECSData>

</SECSMessage>

Conclusion

If you are implementing a GEM interface on an equipment, be sure to allow for messages logging to be enabled/disable and make it easy to download the log files off of the equipment. If you are creating software to use a GEM interface, make sure that you also have a way to enable and disable message logging since this will be a key to troubleshooting any issues that you might run into.

Chapter 13

Gem Control State

The GEM Control State is one of the fundamental E30 GEM requirements. It defines the level of cooperation between the host and equipment and specifies how the operator may interact at the different levels of host control.

In a semiconductor factory, the host or operator may be in control of equipment processing. Having both sides in control of the equipment at the same time poses problems. When one side is in control of the equipment, the other side should be limited in the operations it can perform. For example, if an operator pauses processing, the host should not be allowed to send commands to resume processing or to start a new job. The GEM Control State is provided to prevent these types of issues from occurring.

How does the Control State work?

The Control State provides three basic levels of control. Each level describes which operations may be performed by the host and equipment sides.

Remote

- The host may control the equipment to the fullest extent possible.
- The equipment may impose limits on the local operator's ability to control the equipment, but this is not a requirement of the standard. The host must be capable of handling unexpected commands invoked by the operator at the equipment.
- GEM Remote Commands are used by the host to invoke commands on the equipment.

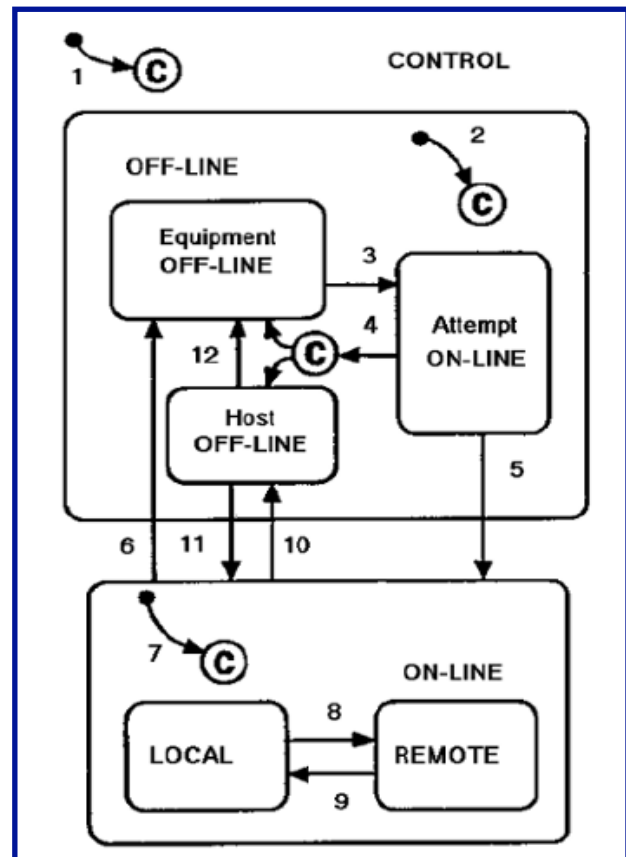


Figure 1: SEMI E30 GEM Control State Model

The Control State Model was designed in a way to give the equipment operator more control over the state machine than the host. This protects the operator from unexpected state changes initiated from the host.

- The equipment operator can choose which Online sub-state is active through the operator interface. The host side cannot choose which Online sub-state is active.
- The equipment side can put the Control State Model into an Equipment Offline state (transition #6). When in this state, the host cannot request to go Online.
- The host side can put the Control State into a Host Offline state (transition #10), but the equipment side could reject this request. When in the Host Offline state, the equipment side can always attempt to go Online by first transitioning into the Equipment Offline state (transition #12) followed by an attempt to go Online (transition #3).

Operator Interface Requirements

The equipment must provide a way of displaying the current Control State to let the operator know who is in control of the equipment.

The equipment must provide a momentary switch to initiate the transition to the Equipment Offline state, and another switch to attempt to go Online from the Equipment Offline state. This may be a hardware switch on the front panel, but is often implemented in software using button controls.

The equipment must provide a discrete two-position switch which the operator may use to indicate the desired Online sub-state (Local or Remote). This may be a hardware switch on the front panel, but is often implemented in software using button controls. If implemented in software, the setting must be saved in non-volatile storage.

Conditional State Transitions

In the Control State Model, transitions #1, #2, #4, and #7 are conditional state transitions. The equipment application must provide a way of configuring which state to transition into. Equipment constants may be used for these configuration settings.

Conditional transitions #1 and #2 determine the initial state of the Control State Model during startup. The configuration that controls these transitions can be set for one of the following states:

- Online
- Equipment Offline
- Attempt Online
- Host Offline

Conditional transition #4 is used to determine which state to transition into after an equipment attempt to go Online fails. The configuration can be set to one of the following states:

- Equipment Offline
- Host Offline

Which Messages are used for Control State?

Message ID	Direction	Description
S1F1	Host <- Equipment	This message is sent to the host when the equipment attempts to go Online (in the "Attempt ON-LINE" state). The host grants permission by sending the S1F2 reply message. The host can deny permission by sending S1F0 or allowing the message transaction to time out.
S1F15	Host -> Equipment	The host sends this message to request a transition from "Host Offline" to Online (transition #11).
S1F17	Host -> Equipment	The host sends this message to request a transition from Online to "Host Offline" (transition #10).

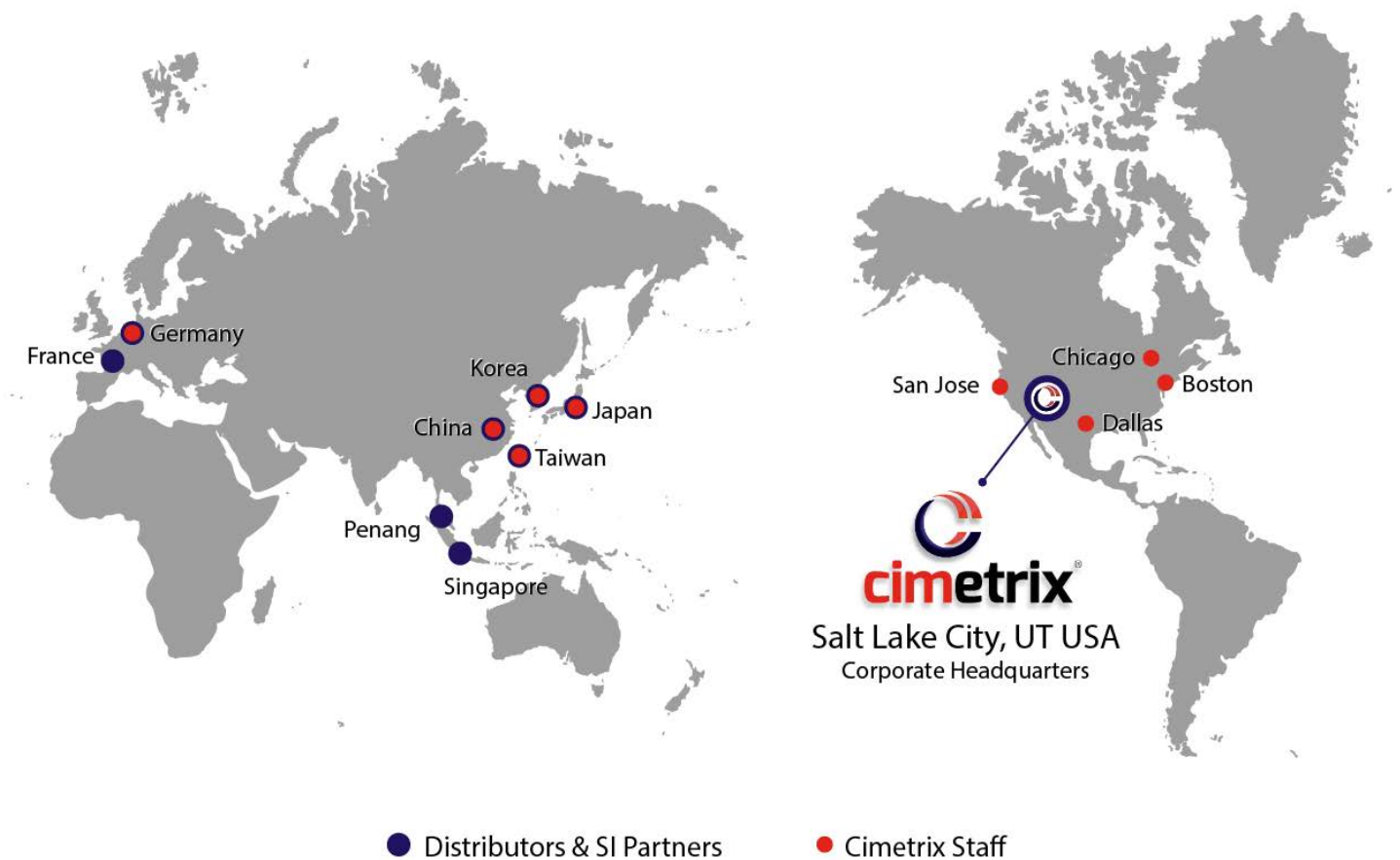
Conclusion

The GEM control state machine is yet one more feature that demonstrates how well the GEM standard is designed for factory automation. It allows the operator or factory to run the equipment, establishes a clear state model to manage which is in charge at all times, and allows the operator to take control at any time.

Chapter 14

Summary

At Cimetrix we hope that you appreciate this collection of topics related to the GEM standard. Clearly the GEM standard is the industry standard that is most suited for all aspects of factory automation, with a broad spectrum of features to empower smart manufacturing and industry 4.0. The GEM standard can be implemented on the simplest and on the most complex of equipment. GEM will continue play a key role in the future of manufacturing in many industries.



www.cimetrix.com/contact

Headquarters
Cimetrix Incorporated
6979 South High Tech Drive
Salt Lake City, UT 84047-3757
USA
Phone: 801.256.6500
Fax: 801.256.6510
Cimetrix.com

Contributing Authors: Alan Weber (Vice President, New Product Innovations), Bill Grey (Distinguished Software Engineer), Brian Rubow (Director of Solutions Engineering), David Francis (Director of Product Management), Derek Lindsey (Product Manager), Jesse Lopez (Client Solutions Engineer), Joe Cravotta (Client Solutions Engineer), Mark Bennet (Client Solutions Engineer), Tim Hutchison (Senior Software Engineer)