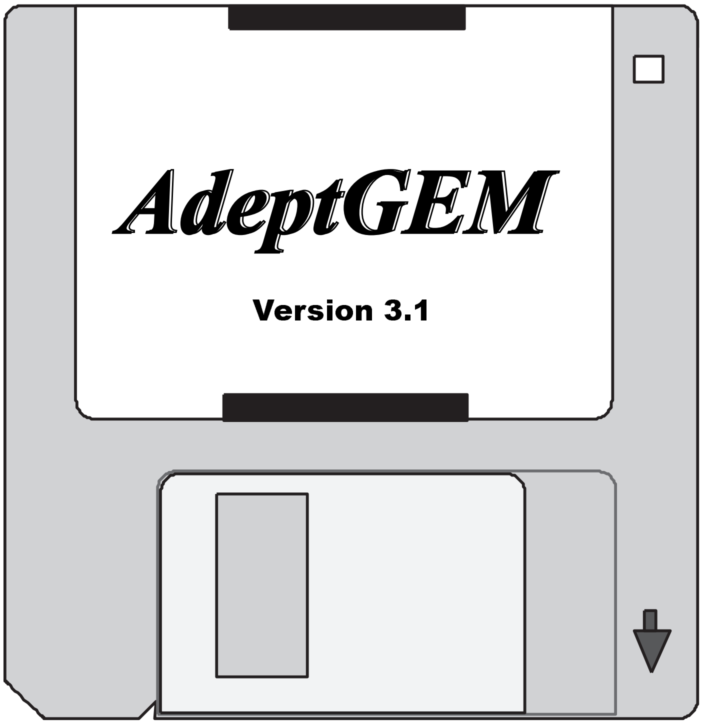
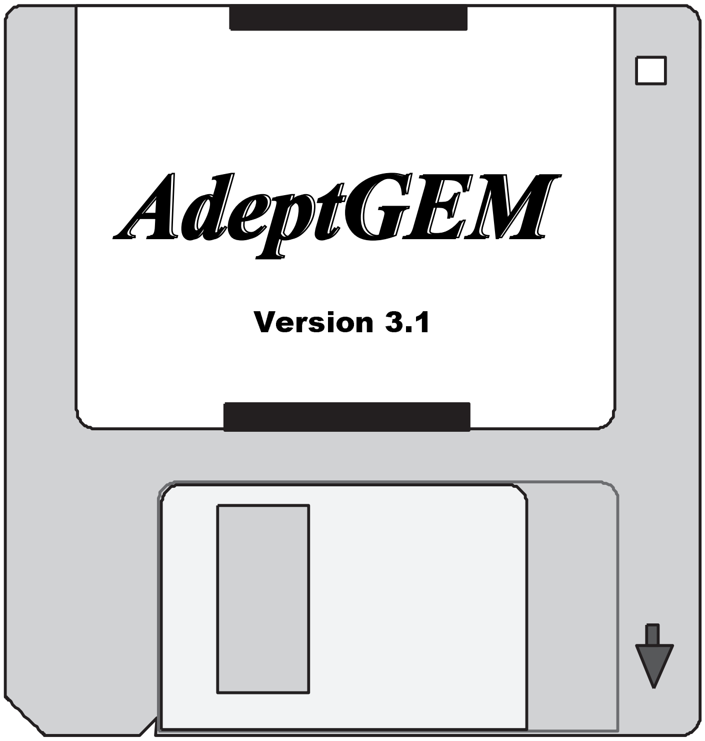
AdeptGEM

User's and Reference Guide



AdeptGEM

User’s and Reference Guide



Part # 00713-01900 Rev. P1
September 1997

adept
technology, inc.

150 Rose Orchard Way • San Jose, CA 95134 • USA • Phone (408) 432-0888 • Fax (408) 432-8707
Otto-Hahn-Strasse 23 • 44227 Dortmund • Germany • Phone (49) 231.75.89.40 • Fax(49) 231.75.89.450
41, rue du Saule Trapu • 91300 • Massy • France • Phone (33) 1.69.19.16.16 • Fax (33) 1.69.32.04.62
1-2, Aza Nakahara Mitsuya-Cho • Toyohashi, Aichi-Ken • 441-31 • Japan • (81) 532.65.2391 • Fax (81) 532.65.2390

The information contained herein is the property of Adept Technology, Inc. and shall not be reproduced in whole or in part without prior written approval of Adept Technology, Inc. The information herein is subject to change without notice and should not be construed as a commitment by Adept Technology, Inc. This manual is periodically reviewed and revised.

Adept Technology, Inc. assumes no responsibility for any errors or omissions in this document. Critical evaluation of this manual by the user is welcomed. Your comments assist us in preparation of future documentation. A form is provided at the back of the book for submitting your comments.

Printed in the United States of America

Table Of Contents

List of Figures

List of Tables

Chapter 1
AdeptGEM Basics

This chapter provides a brief description of the SEMI standards associated with GEM, and how they relate to AdeptGEM. It also provides instructions for installing the AdeptGEM software.

1.1 GEM/SECS Overview

This section provides a brief overview of what the SEMI GEM/SECS specification is, and how the Adept AIM systems supports this specification.

What Is GEM/SECS?

GEM/SECS is a specification developed by the Semiconductor Equipment and Materials International (SEMI) trade association. This standard details the requirements for communication in a GEM/SECS environment. The following specifications are relevant to the AdeptGEM/SECS software:

SEMI E4-91: SEMI Equipment Communications Standard 1, Message Transfer (SECS-I)

SEMI E37-95: High-Speed SECS Message Services (HSMS) Generic Services

SEMI E37.1-95: High-Speed SECS Message Services Single Session Mode (HSMS-SS)

SEMI E5-95: SEMI Equipment Communications Standard 2, Message Content (SECS-II)

SEMI E30-95: Generic Model for Communications and Control of SEMI Equipment (GEM)

These standards can be obtained from SEMI International at (415) 964-5111.

In general, the Adept implementation hides the majority of details of message structuring and packeting from the user. However, you will not be able to effectively integrate equipment into a GEM/SECS environment if you are not familiar with these standards. Throughout this manual we assume that you have this basic familiarity. The capabilities provided in a GEM/SECS environment are extensive, and only by knowing the specification as well as how the AdeptGEM programs work can you design, implement, and debug your equipment.

1.2 GEM/SECS Primer

GEM/SECS provides a method of communication between a host computer system and automation equipment. It specifies the format and allowable content of messages. It also specifies required behaviors of equipment and host during predefined “scenarios” wherein the equipment and host are acting together to perform series of actions.

A Note of Caution

You should note carefully that the primary intent of the specification is to give host equipment a common interface for communicating with many different types of equipment, from simple feeders and conveyors to complex computer controlled equipment such as the Adept MV controller. You may find that, in many cases, the specifications do not allow you to perform actions the way you would like to, and you will be tempted to "bend" the specification to fit your way of operating. However, in the long run, such a strategy will cause problems. The primary benefit of GEM/SECS accrues to the host and the only benefit derived by the equipment is the ability to operate in a host environment that is governed by the GEM/SECS standards. As such, you will be better off working to the intent of the standard and modifying your procedures so they clearly conform to the intent of the standard.

The Levels of GEM/SECS Software

GEM/SECS software can be thought of as operating on three largely independent levels, with each higher level depending on the lower levels but not requiring extensive knowledge of how the lower levels operate.

The Physical Transmission Level

The lowest level is the physical data transmission level. This level describes the physical transmission medium, the format of data transmission, error checking for transmitted data and the handshake between the communicating parties. There are two options for this level, SECS-I and HSMS-SS. SECS-I is a serial communication specification based on RS-232 transmission protocols. HSMS-SS is an ethernet communication specification based on TCP/IP. Other than basic communications parameters such as baud rate, host name, and similar parameters, which you must specify, the AdeptGEM system takes care of all aspects of implementing the standard.

The Message Content Level

The second level is the message content level. This layer describes the format and content of messages than can be exchanged between the host and equipment. These messages are divided into broad classes called streams, and specific operations within those streams called functions.

For example, stream 1 messages deal with the state of the equipment. The stream 1, function 1 message transmits the serial number and ID of the equipment. Again, in general, you will not have to be concerned with the specific format of these messages as the AdeptGEM system will automatically format messages and pass them to the physical transmission level for transmission. You will need to provide your host with specific information about the various stream and function messages. The SEMI standards specify several optional formats and data types for the streams and functions. The documentation in **Chapter 5** provides specific details about how messages transmitted by the AdeptGEM system will be formatted.

The Equipment Behavior Level

The third level is the Generic Equipment Model (GEM), which describes how the host and equipment must interact. The basis of this interaction is a series of "scenarios" that specify which messages must be exchanged under specific conditions. This is the level that you will interact with most. For example, buttons on the GEM control panel establish and terminate communication with the host. You will also be using the AdeptGEM databases to define status variables and reports that can be requested by the host. The GEM compliance issues are handled by the software, but you must specify many variables and events that are particular to your equipment installation and the application that the equipment is performing. AdeptGEM databases predefine most of these variables and you simply have to supply the appropriate values.

This is the level that imposes the most restrictions and requirements on equipment behavior. You should be familiar with this part of the specification so that you can design your AIM implementation to work within the constraints and requirements of the GEM specification.

1.3 AdeptGEM Requirements

In order to install the AdeptGEM system, your equipment must meet the following requirements:

Hardware:

> The system hardware must be adequate to meet the requirements of the devices and software installed on the system. Normally, the AdeptGEM system will not require any hardware upgrades beyond an already adequately configured system.[1] If you are using the SECS-I (serial) option as the communications channel, you must have one free serial port. If you are using the HSMS (ethernet) option you must have an 040 system processor equipped with the AdeptNET hardware.

Software:

> The system must have AIM 3.1 (or later) and a compatible V$^+$ operating system (11.2 or later). If the HSMS option is used, the system must have the AdeptNET TCP/IP option. The "V$^+$ Extensions", "AIM Version 3.1" and "GEM & SECS-I/II Applications" software option licenses must be installed in the Adept controller.

1.4 Installing AdeptGEM

The AdeptGEM system is an add-on AIM module. You should follow the installation instructions for the primary AIM application (for example, MotionWare) that you are using.

Enabling AdeptGEM

To make the AdeptGEM system available for use, open the BASEINI.DB file and set the "GEM Enable" record to ON. The AdeptGEM module is automatically loaded the next time AIM is started.

Adding AdeptGEM to an Existing AIM Installation

You can install the AdeptGEM system to an existing AIM installation simply by copying the AdeptGEM files to the existing installation using the DISKCOPY utility program.

> **NOTE:** If the GEM host will be invoking AIM sequences or V$^+$ routines with the GEM remote control capability (see **section 3.9 on page 33**), you should add the following line to the AIM startup command program (e.g., **lmow( )** in the file LMOW.V2):

```
MC STACK 16 = 20
```

Activating the AdeptGEM Option

If your Adept system was not shipped with the GEM/SECS option already activated, it must be activated. This is a simple matter of typing INSTALL at the system prompt, typing the password that came with the GEM & SECS-I/II Applications license, and then pressing the Enter key. See the description of the INSTALL command in the ***V$^+$ Operating System Reference Guide.***

[1] Systems that are at the limit of available memory or CPU power will probably require the next level upgrade.

1.5 GEM Compliance Statement

The following compliance statement indicates the level of compliance of the baseline AdeptGEM system with the SEMI E30-95 standards.

Table 1-1
GEM Compliance Statement

GEM Compliance Statement		
Fundamental GEM Requirements	**Implemented**	**GEM Compliant**
State Models	■ Yes ❐ No	■ Yes ❐ No
Equipment Processing States	■ Yes ❐ No	■ Yes ❐ No
Host-Initiated S1, F13/F14 Scenario	■ Yes ❐ No	■ Yes ❐ No
Event Notification	■ Yes ❐ No	■ Yes ❐ No
On-Line Identification	■ Yes ❐ No	■ Yes ❐ No
Error Messages	■ Yes ❐ No	■ Yes ❐ No
Documentation	■ Yes ❐ No	■ Yes ❐ No
Control (Operator Initiated)	■ Yes ❐ No	■ Yes ❐ No
Additional GEM Capabilities	**Implemented**	**GEM Compliant**
Establish Communications	■ Yes ❐ No	■ Yes ❐ No
Dynamic Event Report Configuration	■ Yes ❐ No	■ Yes ❐ No
Variable Data Collection	■ Yes ❐ No	■ Yes ❐ No
Trace Data Collection	■ Yes ❐ No	■ Yes ❐ No
Status Data Collection	■ Yes ❐ No	■ Yes ❐ No
Alarm Management	■ Yes ❐ No	■ Yes ❐ No
Remote Control	■ Yes ❐ No	■ Yes ❐ No
Equipment Constants	■ Yes ❐ No	■ Yes ❐ No
Process Program Management	■ Yes ❐ No	■ Yes ❐ No
Material Movement	❐ Yes ■ No	❐ Yes ■ No
Equipment Terminal Services	■ Yes ❐ No	■ Yes ❐ No
Clock	■ Yes ❐ No	■ Yes ❐ No
Limits Monitoring	■ Yes ❐ No	■ Yes ❐ No
Spooling	❐ Yes ■ No	❐ Yes ■ No
Control (Host-Initiated)	■ Yes ❐ No	■ Yes ❐ No

Chapter 2
Fundamental GEM Requirements

GEM compliance is divided into two sections, fundamental requirements and additional capabilities. Fundamental requirements must be implemented by all GEM compliant equipment. Additional capabilities can be optionally implemented as appropriate to the individual equipment. The fundamental requirements are detailed in this chapter. The additional capabilities are detailed in **Chapter 3**.

Many of the capabilities described in this chapter can be categorized in one of three scenarios:

1. Scenarios that occur asynchronously, and that require a primary message to be sent from the equipment (Adept system) to the host. In this case, a separate task detects these scenarios for notification to the host. Various parameters to support these capabilities are specified in AdeptGEM databases.
2. Scenarios that occur synchronously, and that require a primary message to be sent from the equipment (Adept system) to the host. AIM statements are provided to implement this scenario.
3. Scenarios that occur at the host level, and that require some form of response from the equipment (Adept system). These are part of the SECS-II implementation, and are normally transparent to the user. Various parameters to support these capabilities may need to be specified in AdeptGEM databases.

2.1 Introduction

This chapter contains the SEMI required documentation for SEMI E30-95: Generic Model for Communications and Control of SEMI Equipment (GEM) as implemented by AdeptGEM . These standards are described in the publication *Book of SEMI Standard 1995: Equipment Automation/Software 2*.

GEM Compliance Details

The AdeptGEMsystem is in compliance with the protocols defined in SEMI E30-95: Generic Model for Communications and Control of SEMI Equipment (GEM).

The SEMI GEM standard provides compliance specifications for two levels: Fundamental Requirements, and Additional GEM capabilities. The two different levels are described below.

The fundamental requirements are listed below:

- State Models
- Equipment Processing States
- Host-Initiated S1,F13/F14 Scenario
- Event Notification
- On-line Identification

- Error Messages
- Control (Operator-Initiated)
- Documentation

In addition, compliance requires adherence to the portions of the following sections, in the standard, that are applicable to the fundamental GEM requirements:

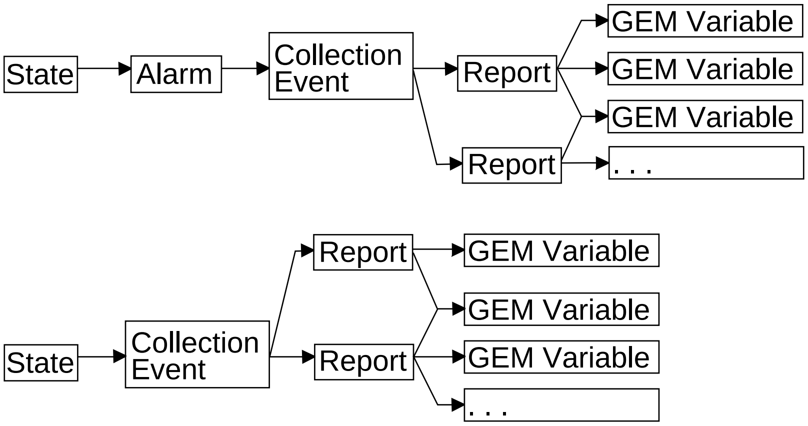
- Variable Data Items
- SECS-II Data Item Restrictions
- Collection Events

The following sections provide detailed information about the support in the AdeptGEM system of the fundamental requirements.

2.2 State Models

SEMI E30-95 Sections 3.0, 3.1, 3.3

Database Usage



Refer to the **“SET_STATE”** statement on **page 84** for details.

Documentation Requirements Equipment must provide state model diagrams and transition tables for all models (the required state models are documented in this manual). All Collection Events and Variable IDs associated with state transitions must be described.

There are two types of state models:

- Models that are required by the SEMI standards and are a basic part of the AdeptGEM module.
- State models that you design and implement as part of an executing sequence.

SEMI Required State Models

There are three required state models:

- The communication state model is described in **"Establish Communications" on page 26** (detailed in Table 3.2 in SEMI E30-95).
- The control state model is described in **"Control (Host-Initiated)" on page 44** (detailed in Table 3.3 in SEMI E30-95).
- The AdeptGEM control state model is described in **"Equipment Processing State Model" on page 18**.

There are three descriptive elements for a state model: The description of states that equipment can be in; the description of events that can trigger changes in equipment states; and the description of the transitions that occur when an event triggers a state change. In order to generate your own state model you will describe all of these events (normally using the descriptive tools described in SEMI E30-95, section 3). You then create the AIM sequence that implements the state model. The sequence will use State records to describe the various events that trigger transitions.

You can make your own state model as follows:

1. Define a variable (in the GEM Variables database) to be used to store a "current" state value. (This variable will be a status variable that can be "seen" by the host through a collection event, status data collection, etc.)
2. Define various state values (in State records in the GEM Items database) for the above variable. "Actions" can be associated with these values that are automatically performed upon a state transition to the value. Actions include:
   - Setting a variable (in the GEM Variables database) to a new value
   - Calling a V$^+$ spawn routine
   - Sending a collection event to the host
   - Setting an alarm
   - Combination of two or more of the above.
3. Define state transitions by including lines in an AIM sequence that change the value of a state variable to a new value.

State Model Example

The following example shows a possible user defined state model in action.

The defined state variable (in the GEM Variables database) is: ProcessMode.

Its possible states (defined in State records in the GEM Items database) are: Action, Analysis.

- A transition to Action state may imply that the part has been removed from the inspection table.
- A transition to Analysis state may imply that a part has been placed on an inspection table.

A possible AIM sequence might be:

```
1.  SET_STATE action
2.  MOVE FROM part1 FROM conveyor TO table
3.  SET_STATE analysis
4.  INSPECT OPERATION part1 OUTPUT sig
```

```
SET_STATE action
IF NOT sig THEN
     MOVE FROM table TO accept.bin
ELSE
     MOVE FROM table TO reject.bin
END
```

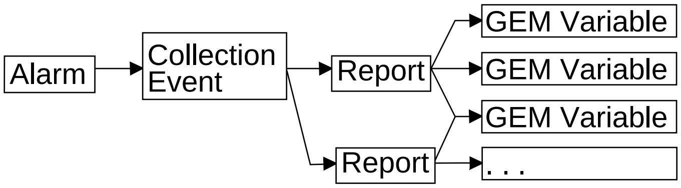
Note that user state models are confined to AIM runtime sequences.

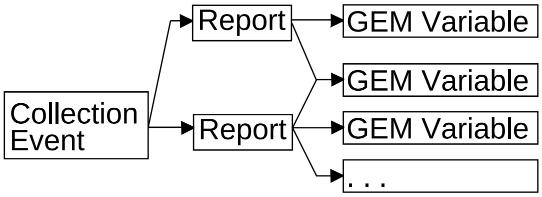
See **Chapter 6** for details on the statements added by the AdeptGEM module.

2.3 Equipment Processing State Model

SEMI E30-95 Sections 3.4

Database Usage





Documentation Requirements The Equipment Processing State model is described in this manual.

The equipment processing state model corresponds to AIM sequence processing states (running, paused, idle, etc.). A state model is maintained for each executing task and separate equipment constants are maintained for the current and previous process states of all tasks. When a state change occurs in the equipment processing state model, DVVAL 9210 is updated with the number of the task that transitioned (see **Table 7-2** for the VIDs used in this state model).

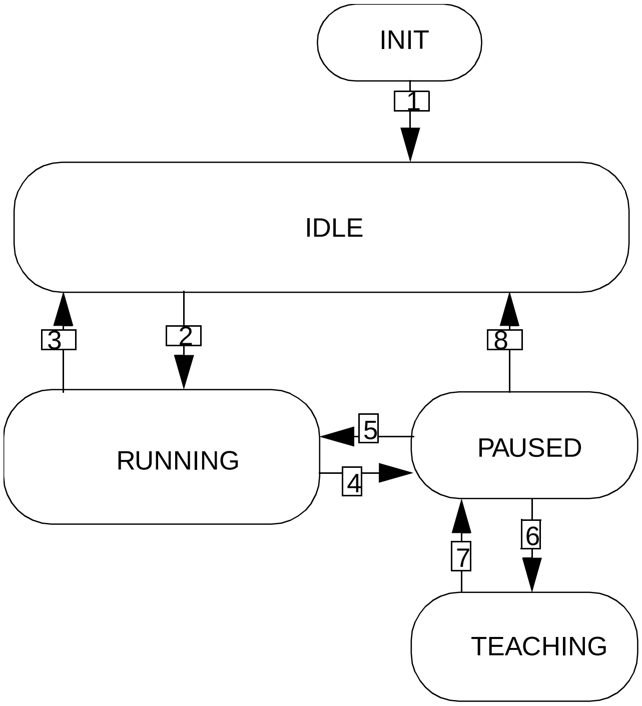


Figure 2-1
Equipment Processing State Model

When AIM is first started, the system will be in the idle state. Equipment processing state model transitions are based on state transitions caused by the AIM control statements.

The system will transition to RUNNING when a START instruction is processed.

The system will transition to PAUSE state if a PAUSE instruction is processed. The system will transition back to the previous RUNNING sub-state when a PROCEED or RETRY instruction is processed. A STOP or PANIC instruction will transition the system to the IDLE state. Pressing **Teach** will cause a transition to the TEACHING state.

A PANIC or STOP instruction will transition the system to the IDLE state.

The current state of the equipment process state for task 0 is stored in status variable #9007 in the GEM Variables database. The previous equipment processing state for task 0 is stored in status variable 9006. The states of the other 26 tasks are stored in status variables 9021 to 9074. The possible states are:

INIT (not recorded, state change occurs before communications enabled)

0. RUNNING
1. TEACHING
2. PAUSED
3. IDLE

The following table shows the possible transitions in the equipment processing state model.

Table 2-2
Equipment Processing State Model Transitions

#	Current State	Trigger	New State	Action	Comments
1	INIT	AIM started	IDLE	None	This state is not recorded since that transition takes place before the system can be in the communicating state.
2	IDLE	Sequence is started	RUNNING	None	
3	RUNNING	A sequence is STOPPED	IDLE	None	
4	RUNNING	A PAUSE instruction is processed	PAUSED	None	
5	PAUSED	A PROCEED or RETRY instruction is pressed	RUNNING	None	An uncleared error will cause an immediate return to PAUSED
6	PAUSED	A TEACH button is pressed	TEACHING	None	
7	TEACHING	The TEACH routine is completed	PAUSED	None	
8	PAUSED	A sequence is STOPPED	IDLE	None	

2.4 Host-Initiated S1, F13/F14 Scenario

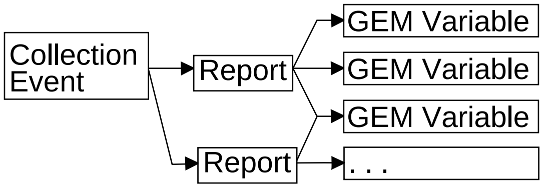
SEMI E30-95 Sections	4.1.5.1
Database Usage	None.
Documentation Requirements	This manual provides all required documentation.

This is a SECS-II level “establish communications” capability in which the host attempts to establish communications with the equipment. These attempts to establish communication are handled automatically by AdeptGEM and the results are displayed on the control panel (see **Figure 3-1**).

2.5 Event Notification

SEMI E30-95 Sections 4.2.1.2

Database Usage



Refer to the **"GENERATE_EVENT"** statement on **page 83** for details.

Documentation Requirements Any Collection Event or VID records created must be documented for the host.

This capability allows the equipment to notify the host when an equipment collection event occurs. Events are stored in the GEM Items database. This section covers both the fundamental event notification requirement and the additional event notification capabilities.

Event notification occurs automatically and does not require operator intervention except to set up the various databases that support event notification. There are three record types used for event notification. The actual event is defined and enabled in a Collection Event record (**page 109**). The Collection Event record specifies Report IDs that are defined in a Report record (**page 112**). The Report record specifies Variable IDs that are defined in the GEM Variables database (**page 86**).

Whenever an enabled Collection Event occurs, a report is automatically generated based on the reports specified in the Collection Event record and the report is sent to the host.

Host Interaction With Collection Events

The host may perform the following actions at any time and without interaction with the equipment operator:

- Define reports
- Link reports to collection events
- Enable/disable collection events
- Delete all defined reports

GEM Requirements

These GEM requirements are specified in Section 4.2.1.2.4 in SEMI E30-95:

1. The collection data will be completely described in the database records of type Collect Event. After creating all your user defined collection events you can use the ASCII export option in the database utilities to export a complete description of all collection events for your various hosts.
2. All CEIDs delivered with AdeptGEM are unique. Any additional CEIDs supplied by you must also be unique.

3. The ☑ **Enabled** check box on the Collection Event menu page allows enabling/disabling of each event.
4. The Report record type allows the user to configure reports associated with a collection event. All variable data is defined in database records which are permanently stored on magnetic media.

2.6 On-line Identification

SEMI E30-95 Sections 4.2.6

Database Usage GEM Variable

Documentation Requirements If you use a model number and software revision different from the default, you must document the meaning of these values.

This is a SECS-II level “establish communications” capability that allows the equipment to identify itself to the host. The host request for identification is handled automatically. The only user activity required is to specify if a user defined software version it to be transmitted. This option is enabled by ECID 9102 in the GEM Variables database and is specified by ECID 9301. (This capability is supported by the S1F1 /S1F2 message scenario.)

2.7 Error Messages

SEMI E30-95 Sections 4.9

Database Usage None.

Documentation Requirements None.

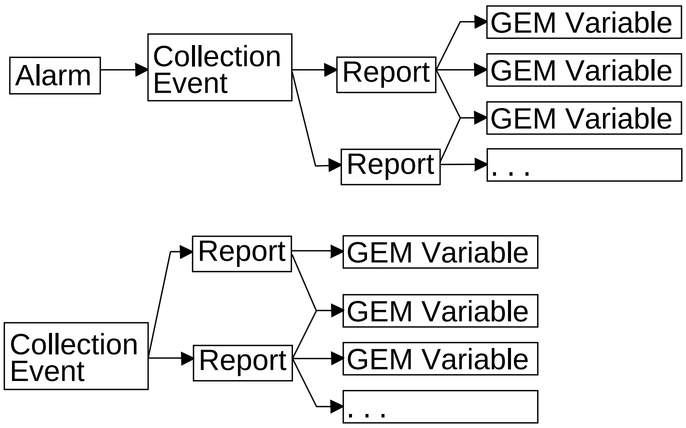
Error messages are implemented as described in Section 4.9.3 in SEMI E30-95, and the SECS-II scenarios described in Section 4.9.5 in SEMI E30-95.

This capability requires no equipment operator intervention or set up.

2.8 Control (Operator-Initiated)

SEMI E30-95 Sections 4.12 — except 4.12.5.2

Database Usage



Documentation Requirements The Control State model is described in this manual.

This requirement is part of the Control State Model, which is described in section 3.14.

2.9 Fundamental Requirements Compliance Issues

Variable Data Items

Required Variable Data Items are supplied with the GEM Variables database and are stored as a special record type (GEM Internal variable type).

SECS-II Data Item Restrictions

The data item restrictions required by section 5.1 “Data Item Restrictions” in the SEMI standards are supported in AdeptGEM.

Collection Events

All collection events are user-defined with a Collection Event record type. For user-defined state models, collection events are triggered by state changes if a collection event has been defined with a State record type. For internal state models, collection events occur on state changes. The collection event IDs are hard coded, but the collection records for those IDs can be modified by the user.

Collection events can also be triggered explicitly by an AIM statement.

Chapter 3
Additional GEM Capabilities

This chapter details the implemented GEM capabilities that are listed in section 8.2 of SEMI E30-95.

Many of the capabilities described in this chapter can be categorized in one of three scenarios:

1. Scenarios that occur asynchronously, and that require a primary message to be sent from the equipment (Adept system) to the host. In this case, a separate task detects these scenarios for notification to the host. Various parameters to support these capabilities are specified in AdeptGEM databases.
2. Scenarios that occur synchronously, and that require a primary message to be sent from the equipment (Adept system) to the host. AIM statements are provided to implement this scenario.
3. Scenarios that occur at the host level, and that require some form of response from the equipment (Adept system). These are part of the SECS-II implementation, and are normally transparent to the user. Various parameters to support these capabilities may need to be specified in AdeptGEM databases.

3.1 Introduction

This chapter contains the SEMI required documentation for SEMI E30-95: Generic Model for Communications and Control of SEMI Equipment (GEM) as implemented by AdeptGEM. These standards are described in the publication *Book of SEMI Standard 1995: Equipment Automation/ Software 2*.

GEM Additional Capabilities Compliance Details

The AdeptGEM system is in compliance with the protocols defined in SEMI E30-95: Generic Model for Communications and Control of SEMI Equipment (GEM).

- Establish Communications
- Dynamic Event Report Configuration
- Variable Data Collection
- Trace Data Collection
- Limits Monitoring
- Status Data Collection
- Alarm Management
- Remote Control

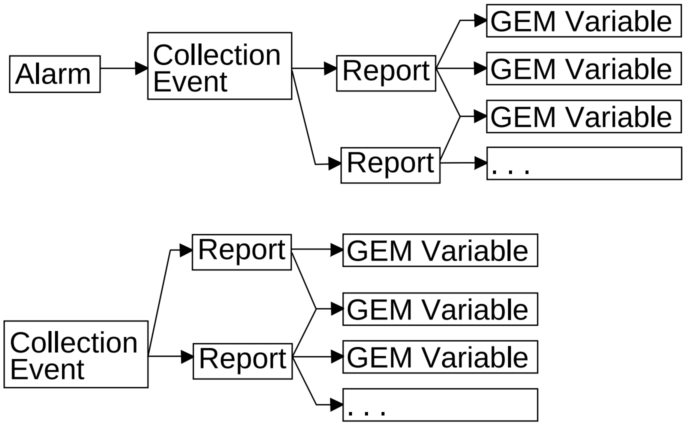
- Equipment Constants
- Process Program Management
- Material Movement
- Equipment Terminal Services
- Clock
- Control (Host-Initiated)

The following sections provide detailed information about the support in the AdeptGEM system for these GEM capabilities.

3.2 Establish Communications

SEMI E30-95 Sections 4.1, 3.2

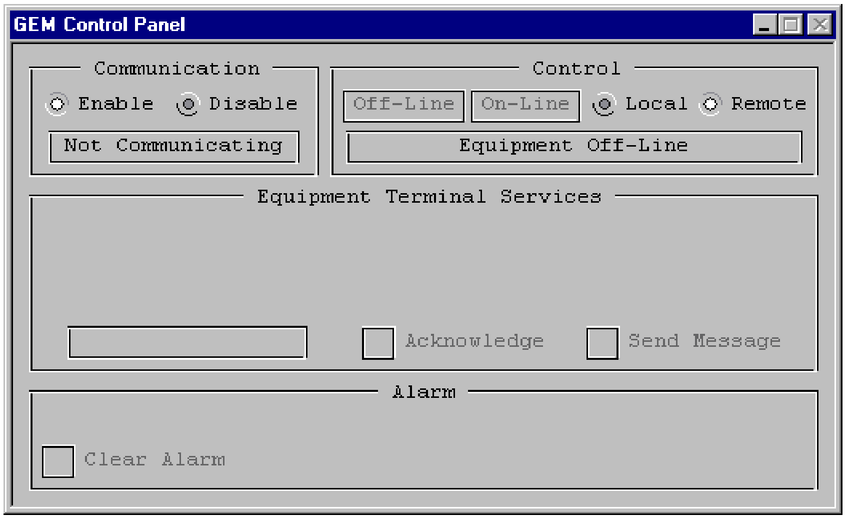
Database Usage



Documentation Requirements None.

Communications are established in accordance with the GEM Communications State model. The GEM Control Panel shown below shows buttons and indicators that implement the Communications State Model. To display the GEM Control Panel:

Show ➡ GEM Control Panel



GEM Control Panel

Communication

○ Enable ● Disable

Not Communicating

Control

Off-Line On-Line ● Local ○ Remote

Equipment Off-Line

Equipment Terminal Services

□ Acknowledge □ Send Message

Alarm

□ Clear Alarm

Figure 3-1
GEM Control Panel, Establish Communication Options

Selecting ● **Enabled** will start the scenario that attempts to establish communication with the host. If the attempt is successful, the new communicating state will be shown at the bottom of the "Communication" group.

Selecting ● **Disabled** will start the scenario that attempts to halt communication with the host. If the attempt is successful, the new communicating state will be shown at the bottom of the "Communication" group.

Before communication can be successfully enabled, the communication channel must be properly installed and configured and the host must be ready to establish communications. All the configuration options for the serial or ethernet channel as well as the GEM parameters are set in the GEM Variables database.

The equipment constants that define the communication configuration and GEM parameters are included in **Table 7-2**. They are also summarized in Tables **4-1** through **4-3**.

VID 9300 specifies the serial channel ID for SECS-I or the ethernet ID for HSMS. The possible values for a serial channel are: LOCAL.SERIAL:1, LOCAL.SERIAL:2, SERIAL:1, SERIAL:2, and SERIAL:3. See the AdeptNET user's guide for details on proper ethernet channel names.

AdeptGEM Debug Mode

AdeptGEM provides a window that traces the message traffic and success state of message transmission. The window is displayed whenever communications is enabled and EC 9105 is set to enable the windows. (This EC is checked only at start-up, so a restart is required to change the debug mode.) A log of the last 64 debug messages is stored in EC 9010.

GEM Requirements

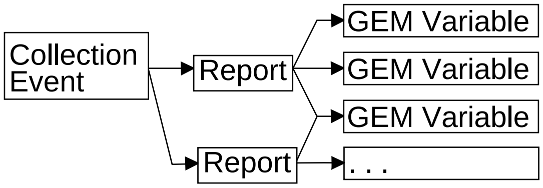
These GEM requirements are specified in Section 4.1.4 of SEMI E30-95:

1. The communication state model is fully implemented as described above.
2. The EstablishCommunicationsTimeout equipment constant is stored as ECID 9100 in the GEM Variables database.

3.3 Dynamic Event Report Configuration

SEMI E30-95 Sections 4.2.1.2

Database Usage



Documentation Requirements Any Collection Event or VID records created must be documented for the host.

This capability is implemented as specified in the SEMI standards. It allows the host to specify which Collection Events it wants enabled. It is largely transparent to the local operator and equipment. The only requirement is that the operator define the Collection Events and associated reports as described in section 2.5.

GEM Requirements

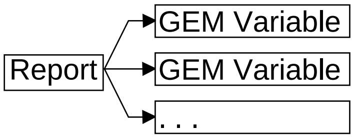
These GEM requirements are specified in Section 4.2.1.2.4 of SEMI E30-95:

1. The variable data will be completely described in the database records that hold the variables (GEM Variables database). After creating all your user defined SVs and ECs you can use the ASCII export option in the database utilities to export a complete description of all IDs for your various hosts.
2. All VIDs delivered with AdeptGEM are unique. Any additional VIDs supplied by you must also be unique.
3. All variable data is available for report definition. See **"The Report Record Type" on page 112** for details on creating reports on event data collection.
4. All report definitions, report-to-event links, and enable/disable status is defined in database records which are permanently stored on magnetic media.

3.4 Variable Data Collection

SEMI E30-95 Sections 4.2.2

Database Usage



Documentation Requirements You must document any Reports or Variables that you create.

This capability is implemented as specified in the SEMI standards. It allows the host to request that specific reports be generated and transmitted. It is largely transparent to the local operator and equipment. The only requirement is that the operator define the reports with Report record types (page 112) and the corresponding VIDs in the GEM Variables database (page 87).

3.5 Trace Data Collection

SEMI E30-95 Sections 4.2.3

Database Usage

Documentation Requirements You must document any Status Variable IDs that you create.

V^{+} Task Usage Limits monitoring trace data collection.

This capability is implemented as specified in the SEMI standards. It allows the host to request the values of defined status variable (SVs). These requests have a time constant associated with them and are transmitted at intervals specified by the time constant. It is largely transparent to the local operator and equipment. The only requirement is that the operator define the SVs in the GEM Variables database (section 7.1).

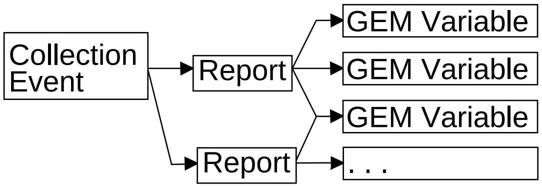
> **NOTE:** As noted in section 4.2.3.4 of SEMI E30-95 there is a potential problem with the message that transmits the SV report. The SEMI standard allows only a single block message, however, the allowed format for SVs can be larger than a single block. In order to conform to the standard the host must not request reports that will exceed a single block size. (In many cases, hosts have been designed to ignore this restriction and accept multi-block messages. AdeptGEM can generate a multi-block messages if this is the case.)

GEM Requirements

These GEM requirements are specified in Section 4.2.3.4 of SEMI E30-95:

1. AdeptGEM uses the V+ internal timers for triggering the periodic sampling.
2. Up to four concurrent traces can be maintained.
3. Any SVID in the GEM Variables database can be included in a trace data collection report. See the above NOTE for details on the single block restriction.

3.6 Limits Monitoring

SEMI E30-95 Sections	4.2.4
Database Usage	 Collection Event Report Report GEM Variable GEM Variable GEM Variable . . .
Documentation Requirements	You must document any Collection Events or Variables that you create.
V+ Task Usage	Limits monitoring trace data collection.

Limits monitoring allows the host to request that a value from a record in the GEM Variables database be monitored to see if it violates a set of defined limits. The host specifies the limits and the polling interval for the monitoring. The only actions required by the equipment are:

- Create a SVID in the GEM Variables database that will contain the value that is being monitored.
- Provide a way (sequence statement or V+ routine) to update the Status Variable.
- Indicate which Collection Event is to be generated when the value crosses a limit. This event is specified in the "Limit Mon. Event" data box on the GEM Variables database menu page (**Figure 7-6**).
- Define the reports used by the Collection Event. (These reports may also be defined by the host.)

Once these records are available, limits monitoring is transparent to the user.

GEM Requirements

These GEM requirements are specified in Section 4.2.4.4 of SEMI E30-95:

1. Seven limits may be monitored per variable.
2. The CEID for the monitored variable is specified in the GEM Variables database record that specifies which value is to be monitored.
3. All variable data for the monitored value is defined in database records that are permanently stored on magnetic media. The monitoring parameters supplied by the host are stored in a data file whenever the AIM system is shut down.

4. The limits monitoring algorithm makes all SEMI required checks for valid limits monitoring requests received from the host.
5. The SVID records in the GEM Variables database can be completely documented by the user. These records can be exported in ASCII format to create host required documentation.

3.7 Status Data Collection

SEMI E30-95 Sections	4.2.5
Database Usage	GEM Variable
Documentation Requirements	You must document any Status Variable IDs that you create.

This capability is implemented as specified in the SEMI standards. It allows the host to request the values of defined status variable (SVs). It is largely transparent to the local operator and equipment. The only requirement is that the operator define the SVs in the GEM Variables database (section 7.1).

GEM Requirements

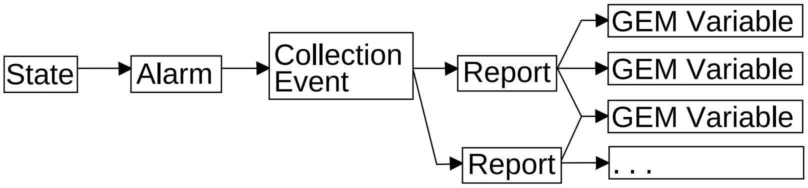
These GEM requirements are specified in Section 4.2.5.4 of SEMI E30-95:

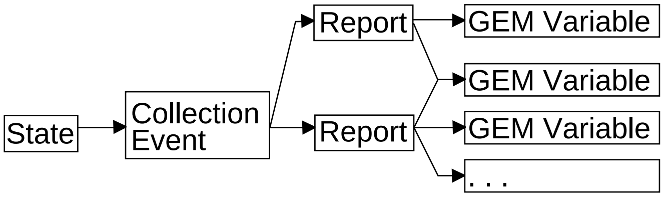
1. Built-in, unique SVIDs are listed in Table 7-2 and are provided in the standard databases delivered with AdeptGEM. Any additional SVIDs created by the user must be unique.
2. All SVIDs in the GEM Variables are available for data collection.
3. All SVs defined by AdeptGEM will contain valid data when the SV is transmitted to the host. The user must guarantee valid data for any SVs created by the user.

3.8 Alarm Management

SEMI E30-95 Sections 4.3

Database Usage





Refer to the **"GENERATE_ALARM"** statement on **page 83** for details.

Documentation Requirements You must document any defined Alarms, Collection Events and VIDs associated with the Alarms.

V+ Task Usage Limits monitoring trace data collection.

See **"The Alarm Record Type" on page 108** for details on defining an alarm.

Alarms can to be set in three ways:

1. When a state in a user-defined state model is entered, the user can indicate that an alarm be trigger. The alarm to be triggered is defined by a State record type. See **"The State Record Type" on page 113** for details on specifying an alarm for a state transition.
2. An AIM sequence may explicitly turn on an alarm. See **"GENERATE_ALARM" on page 83** for details on setting an alarm using a statement.
3. A GEM variable can be monitored for a transition to an unsafe value. If this occurs, an alarm is triggered. (The system must be in the communicating state for this option.) See **"Automatic Alarm Detection"** below.

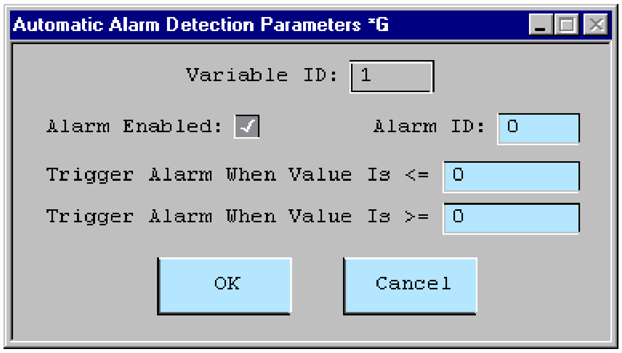
After an alarm is set, the host will be notified (if reporting is enabled). Additionally, the alarm will be displayed on the GEM Control Panel (see **Figure 3-1**). If desired, an AIM sequence can be run (configured in the Alarm record type).

When the operator presses **Clear Alarm** on the control panel (see **Figure 3-1**), it is assumed that the error condition has been corrected. The alarm is cleared automatically, and a clear acknowledgment is sent to the host (if reporting is enabled). If the alarm was initially triggered by a monitored variable, it is possible that the variable could still be in a bad condition. The alarm will not be triggered again unless there is a transition to a good zone, and then back to a bad zone. Consider this carefully when designing alarms. An AIM sequence may be the only way to ensure proper factory safety requirements; on the other hand, this simple model may work well for things like monitoring an input signal and producing an alarm if it is turned on (or off).

NOTE: During AIM startup, any alarm that is found set (i.e., from a previous session) is automatically cleared (without notice).

Automatic Alarm Detection

Automatic alarm detection is setup by pressing **Auto Alarm Detection** on the GEM Variables database record that specifies the alarm variable (see **Figure 7-6**).



Automatic Alarm Detection Parameters *G

Variable ID: 1

Alarm Enabled: ☑

Alarm ID: 0

Trigger Alarm When Value Is <= 0

Trigger Alarm When Value Is >= 0

"Variable ID:" shows the VID from the GEM Variables database that has been selected for monitoring. In Alarm ID, specify a defined alarm ID from the Alarm database (this alarm will be triggered if the limit is exceeded). Set the upper and lower limits for the alarm and then select Alarm Enabled. This VID will now be monitored automatically and compared with the limits specified. If the limits are reached, the actions specified in the Alarm database record for the Alarm ID will be triggered.

GEM Requirements

These GEM requirements are specified in Section 4.3.4 of SEMI E30-95:

1. All alarms and appropriate trigger(s) are user-defined with an Alarm record type as described above.
2. The enable/disable states of all alarms as well as the report definitions and collection events are stored in AIM databases on magnetic media.
3. The alarm database contains fields for a description, ALID, ALTX, ALCD and two CEIDs.
4. Enabled alarm reports will be sent prior to corresponding enabled event reports.

3.9 Remote Control

SEMI E30-95 Sections 4.4

Database Usage None.

Documentation Requirements You must document the behavior of any defined remote commands and the usage of any parameters associated with the commands.

Remote control can be used by the host to:

- Execute a predefined command (PANIC or SPEED)
- Execute a V^{+} program
- Execute an AIM control sequence

To determine how the host request is processed, the AdeptGEM software looks first for a match with a predefined command, second for a match with a V^{+} program name, and third for a sequence spawn. See the description of the routine **gm.remote.cmd( )** on **page 161** for details.

Status of the AdeptGEM System

There are three V^{+} global variables that store the status of the AdeptGEM system:

gm.comm — TRUE if the equipment is in the COMMUNICATING communications state.

gm.online — TRUE if the equipment is in the COMMUNICATING communications state and the ON-LINE control state, or if the equipment will go into the ON-LINE control state when communication is enabled.

gm.remote — TRUE if the equipment is in the COMMUNICATING communications state and the on-line REMOTE control state, or the equipment will go into the REMOTE control state when the equipment goes on-line.

Remote Control and AIM Control Sequences

AIM control sequences and statements allow you to perform the SELECT, PAUSE, PROCEED, and STOP operations for a standard sequence. This capability provides a general purpose facility for creating remote commands that can affect sequences singly or in groups.

Control sequences are generally stored in the “control” module for each AIM application (MOWCTL.MOD for MotionWare, VWCTL.MOD for VisionWare, and PCBCTL.MOD for AIM PCB). This module is automatically loaded when AIM is started, so any sequences created in this module will be available at start-up.[1]

These sequences will execute in the main AdeptGEM interface task, so you do not have to specify a task or be concerned with whether a task is available to execute a remote command. However, the interface task will be tied up while the control sequence is executing, so these sequences should have as short an execution time as possible.

Host Remote Commands

There are two predefined remote commands, PANIC and SPEED that perform the same functions as these two features on the control panels.

- Panic button

```
L, 2
   1. "PANIC"
   2. L,0
```

[1] The default control module can be changed with an initialization record:
Setup ➡ Initialization Data ➡ select “mowini” ➡ change “control sequence module”

- Change robot speed

```
L, 2
   1. "SPEED"
   2. L, 2
      1. L, 2
         1. "VALUE"
         2. <New speed value>
      2. L, 2
         1. "TASK"
         2. <Task to set speed for>
```

Once a control sequence or V$^+$ program has been created, it can be invoked by the host sending an S2F41 message with the sequence or program name specified as the RCMD parameter. Parameters for the sequence or program can be included in the message. See **Chapter 5** for more information on the message format.

NOTE: If the GEM host will be invoking AIM sequences or V$^+$ routines with the GEM remote control capability, you should add the following line to the AIM startup command program (e.g., **lmow( )** in the file LMOW.V2):

```
MC STACK 16 = 20
```

Host Control of Equipment

The **On-Line** / **Off-Line** buttons on the GEM Control Panel are the only way to transition between equipment on-line and equipment off-line (the host cannot request a transition from the equipment off-line state). A host on-line request will result in a transition only when the equipment is in the host off-line state.[1]

GEM Requirements

The SEMI standard has several specific requirements for remote commands:

1. The START and STOP commands must be implemented as defined by the SEMI standard. Since implementing the actual requirements will be different for each installation, it is up to the user to create these two commands and to ensure that they are SEMI compliant.
2. The following commands are optional. However, if they are implemented, they must match the SEMI description for the command: PP_SELECT, PAUSE, RESUME, and ABORT.
3. Remote commands may be sent in mixed letters.

These GEM requirements are specified in Section 4.4.4 of SEMI E30-95:

Control State Model Restrictions

The Control State model defines the level of cooperation between the host and equipment. It also specifies how the operator may interact at the different levels of host control. The host and equipment are restricted from executing certain programs. See the paragraphs below for details.

[1] Note that “equipment off-line” does not physically break the communication link with the host. The host can still transmit messages and the equipment will respond with the appropriate message indicating that communication is disabled.

Also, see section **3.10 “Equipment Constants”,** subsection **”GEM Requirements”**, item **2** for details.

The host shall have the following capabilities and restrictions when the LOCAL state is active:

- The host shall be prohibited from the use of remote commands that cause physical movement or which initiate processing. During processing, the host shall be prohibited from the use of any remote command that affects that process.

When the on-line REMOTE state is active, the host may operate the equipment to the full extent available through the communications interface. However, the operator will be restricted in specific capabilities. These restrictions should be configurable so that the equipment may be set up to allow the operator to perform necessary functions without contending with the host.

The statement RUN_CHECK (see **section 6.1 on page 83**) can be used in a sequence program to verify the current state of the equipment as shown in the following example:

```
RUN_CHECK allowed = MOTION_OK
IF allowed THEN
...
ELSE
   MESSAGE “Motion Not Allowed!”
END
```

3.10 Equipment Constants

SEMI E30-95 Sections	4.5
Database Usage	GEM Variable
Documentation Requirements	You must document any Equipment Constant IDs that you create.

Equipment constants provide a way to store values for equipment parameters. These values control various aspects of equipment behavior (i.e., how many widgets to build, what parts to use for a widget assembly, etc.). Equipment constants are stored in the GEM Variables database (section **7.1**).

This capability is implemented as specified in the SEMI standards. It is largely transparent to the local operator and equipment. The only requirement is that the operator define the equipment constants for any required constants that are not already defined in the GEM Variables database.

Built-in equipment constants are summarized in **Table 7-2** and are referenced throughout this manual. User-defined ECs can be added to the GEM Variables database.

GEM Requirements

These GEM requirements are specified in Section 4.5.4 of SEMI E30-95:

1. Equipment constants (ECs) are stored in AIM databases on magnetic media.
2. The equipment constant is a constant when viewed from the equipment when the control state is on-line REMOTE.

- The host can change the EC anytime except for Process Related restrictions (see table below).
- The equipment cannot change the EC when on-line REMOTE
- The equipment can change the EC when off-line

The host shall have the following capabilities and restrictions when the LOCAL state is active:

- During processing, the host shall be prohibited from modifying any equipment constants that affect that process. Other equipment constants shall be changeable during processing, The host shall be able to modify all available equipment constants when no processing is in progress.

CAUTION: It is the customer’s responsibility to make sure that the equipment is in a “safe” condition before setting equipment constants. To help handle most cases the user can mark a variable as being “process related” or “not process related”. The table below shows the relationship between the Control mode, the process state, and EC modification.

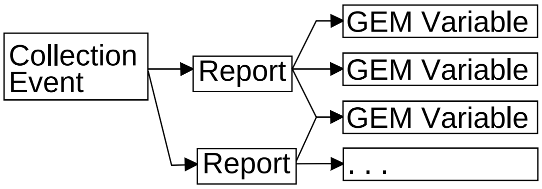
Control	**Process Related Setting**	**Process State**	**Equipment Constant Modification**
LOCAL	Process related ☑	Running	No modification is allowed.
REMOTE	Process related ☑	Running	May be operator modified depending on the setting of the REMOTE mode configuration category. This is set in the GEM Variables database.
LOCAL	Process related ☑	Not running	Modification is allowed.
REMOTE	Process related ☑	Not running	Modification depends on the REMOTE mode configuration category setting for *non-process* related variables.

3. The equipment must provide a collection event to alert the host whenever an equipment constant is changed by the operator. (See the built-in collection events listed in **Table 7-3**. The collection event name is “Operator Equipment Constant Change”. The number is 9200.)

3.11 Process Program Management

SEMI E30-95 Sections 4.6

Database Usage



Documentation Requirements Name of program available for uploading.

Process program management is performed as specified by the SEMI standards.

What is a Process Program?

The SEMI standards define a process program in only a very general manner, and this description does not fit a sophisticated multi-tasking environment like AIM. In AIM, in order to define a workcell implementation, you create two types of resources; logical resources and data resources. The logic resources are the AIM sequences, and an arbitrary number of sequences can be used in an implementation. The data resources are AIM databases, and an arbitrary number of databases can be accessed by an executing sequence or group of sequences. A process program in the AdeptGEM system is defined as an AIM Resource Module. When you request information on a process program, the AdeptGEM system will search the module directory and return information on the specified resource module.

Process Program Directory

All requests related to process program management will use the directory path specified in EC 9302, which specifies the path that will be used for all process program management requests (delete, directory, transfer, etc.).

Process Program Operations

Host requests to upload, download or delete a process program can specify a resource module or a disk file. For those operations, a PPID that does not contain a period character (".") is interpreted as the name of a resource module; a PPID that does contain a period character (".") is interpreted as the name of a disk file.

The host can upload any resource module or disk file by initiating the proper process program management upload scenario. The host can download as many resource modules and disk files as can be stored on disk. Disk space availability is not checked by the AdeptGEM system. The transfer is handled without any operator interaction.

The equipment user can request upload/download of a resource module or disk file with the menu options:

Setup ➡ GEM: Upload To Host and **Setup ➡ GEM: Download From Host**.

When the host requests that a resource module or disk file be deleted, it will be removed from disk without operator intervention or verification.

> **CAUTION:** SEMI requires that if the host sends a delete request with no name, the equipment must delete all process programs in the directory. This is done without operator intervention.

When the host requests an Equipment Process Program Directory (EPPD), the names of all the resource modules in the directory specified by EC 9302 will be transmitted. Thus, the list may include loaded as well as unloaded modules.

GEM Requirements

These GEM requirements are specified in Section 4.6.4 of SEMI E30-95:

1. All resource modules can be created, modified and deleted using the standard AIM interface. Resource modules can be deleted and transferred using the SEMI process program management capabilities. Resource modules cannot be modified by the host.
2. Host requests to upload, delete, and list resource modules on disk are handled automatically. Programs downloaded from the host are written to disk.

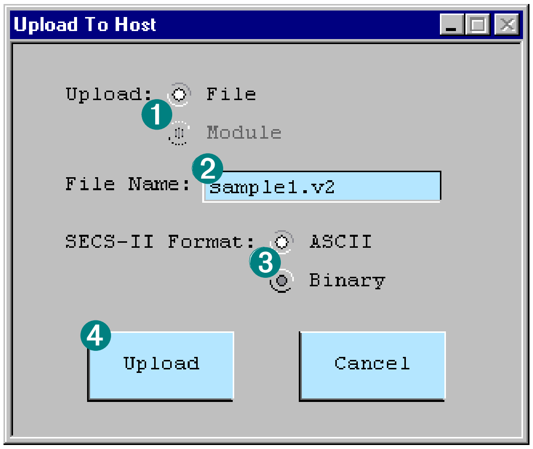
> **CAUTION:** SEMI section 4.6.3 requires that “If a process program exists with the same PPID as the one given in the SECS-II message, the old process program must be replaced.” This means that any existing resource modules or disk files with the same names will be overwritten without warning.

3. The equipment can store as many process programs as memory will allow. The logic resources associated with these process programs will not be modified by execution. However, the data resources may be modified during execution (updating counters in the variable database, for example).
4. The files in an AIM Resource Module are transmitted in a single file with a specific format. When this file is downloaded, it must be in the same format. See **“Operations With Standard Disk Files”** for details on transferring standard files.
5. Downloaded resource modules are verified when they are received from the host. (Standard disk files are not verified.)
6. Only unformatted process programs are supported.
7. The maximum size resource module that can be transmitted is limited by the size of the communication buffer specified by EC 9154 (commbufsize). The initial setting for this buffer is 28Kbytes.

Equipment Process Program Requests

The equipment user can request an upload or download of a resource module or disk file. To request that a resource module or file be uploaded:

Setup ➡ GEM: Upload To Host



Upload To Host

Upload: ○ File ○ Module

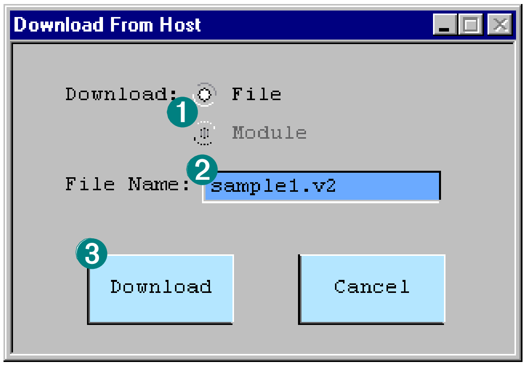
File Name: sample1.v2

SECS-II Format: ○ ASCII ○ Binary

1. Select the type of transfer (item ❶).
2. Enter the name of the resource module or disk file (item ❷).
3. For a file transfer, select the data item format (● **ASCII** or ● **Binary** identifier) to be used for the transfer (item ❸). (Resource modules are always transferred as binary data.)
4. Press **Upload** (item ❹) to begin the transfer.

To request that a program or resource module be downloaded:

Setup ➡ GEM: Download From Host



Download From Host

Download: ○ File ○ Module

File Name: sample1.v2

1. Select the type of transfer (item ❶).
2. Enter the name of the resource module or disk file (item ❷).
3. Press **Download** to begin the transfer (item ❸).

If the upload/download is successful, these windows are closed automatically.

Operations With Standard Disk Files

As an additional capability, the process program upload, download, and delete capabilities can be used with standard disk files. To differentiate between a request for a Resource Module and a standard file, add the file extension to the request (e.g., JOB1=Resource Module; JOB1.V2=standard file). This additional capability is supported only for file upload, download, and delete. No verification or validation is performed on transfers of standard files and you can not get a directory of disk files.

3.12 Equipment Terminal Services

SEMI E30-95 Sections 4.8

Database Usage GEM Variable

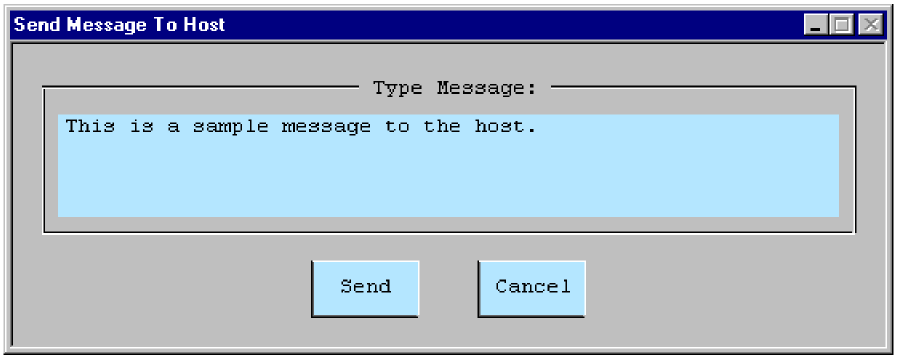
Refer to the **"SEND_HOST_MSG"** statement on **page 84** for details.

Documentation Requirements None.

This capability allows the host and equipment to display messages on each other's terminals.

The equipment operator can send a terminal message to the host by:

Setup ➡ GEM: Message to Host



This is a sample message to the host.

Figure 3-2
Message Window

Type the message to be sent into the message area and the press **Send** to send the message. Press **Cancel** to abort the message.

A message can also be sent by pressing **Send Message** on the GEM Control Panel, or by including the statement SEND_HOST_MSG in a sequence.

If a message is received from the host, it will be displayed in the "Equipment Terminal Services" section on the GEM Control Panel (see **Figure 3-1**). There are two options:

Press **Acknowledge** to return the acknowledgment Collection Event.

GEM Requirements

These GEM requirements are specified in Section 4.8.4 of SEMI E30-95:

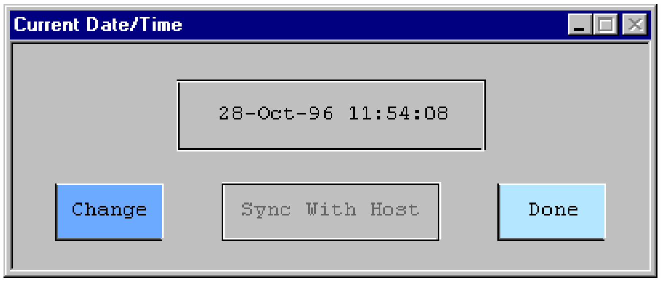
1. Any new terminal display message will overwrite an unrecognized message at the same terminal.
2. The equipment will display at least 160 characters. The characters may be displayed on more than one line.
3. Messages received from the host will be displayed on the control panels.
4. The message will be displayed on the control panel when the message is received to alert to the operator that there is a message from the host.
5. An **Acknowledge** button on the control panel will notify the host the message was acknowledged.
6. The fixed maximum size of a message that can be received from the host will be 240 characters. Longer messages will be truncated.
7. As described above, the GEM control panel will accept message text and generate an appropriate SECS-II message.
8. Multi-block messages are supported.
9. A zero length terminal message will erase any previous message from the host (at which point the host will not expect an acknowledgment).

3.13 Clock

SEMI E30-95 Sections	4.10
Database Usage	None.
Documentation Requirements	None.

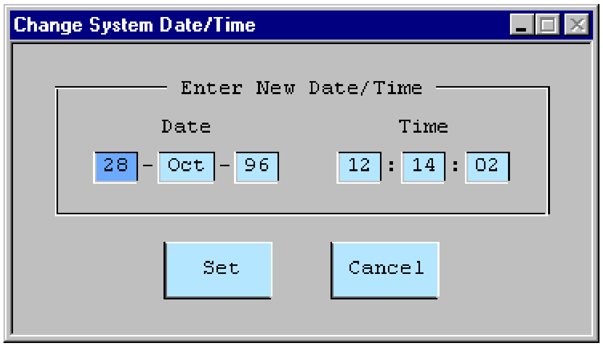
The clock requirement is implemented as defined by SEMI. The optional scenario where the operator requests a time value from the host is implemented. The operator of the equipment can request host time as described below. To display the AdeptGEM clock:

Setup ➡ GEM: Time



28-Oct-96 11:54:08

Press **Sync With Host** to automatically synchronize the equipment clock with the host clock.[1] Press **Change** to display the following window from which you can set the current date and time.



Enter New Date/Time

Date: 28 - Oct - 96

Time: 12 : 14 : 02

GEM Requirements

These GEM requirements are specified in Section 4.10.4 of SEMI E30-95:

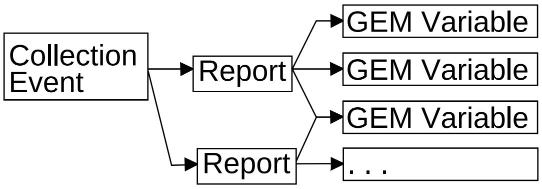
1. The resolution of the time reference exceeds the speed at which an individual collection event can be recorded, thus all events will have different time stamps.
2. The resolution of the time reference is based on the V$^+$ one millisecond timer.
3. Centisecond values are assigned based on the actual value of the clock. Unresolvable simultaneous events will be given the actual time value of their occurrence.

[1] The equipment must be online and communicating.

3.14 Control (Host-Initiated)

SEMI E30-95 Sections 4.12.5.1

Database Usage

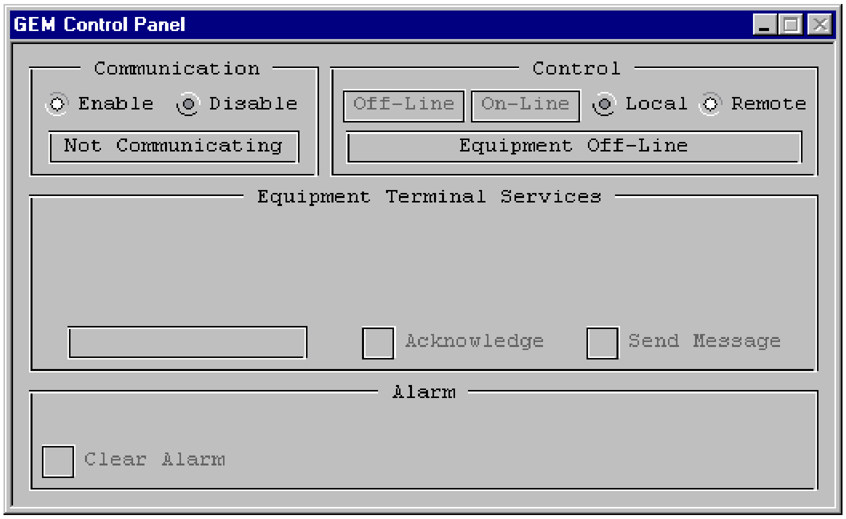


Documentation Requirements None.

This section describes the Control state model that implements both the Operator Initiated Control requirement and the Host Initiated control capability.

Changes to the control state are made at the equipment from the GEM Control Panel. Host changes to the control state are made automatically and reflected on the control panel. This control panel must be used when SEMI compliant operation is required. To display the GEM Control Panel:

Execute ➡ GEM Control Panel


GEM Control Panel

Communication

○ Enable ● Disable

Not Communicating

Control

Off-Line On-Line ● Local ○ Remote

Equipment Off-Line

Equipment Terminal Services

□ Acknowledge □ Send Message

Alarm

□ Clear Alarm

Figure 3-3
Control State Model Options

Before changes can be made to the control state model, communications must be enabled.

Pressing **Off-Line** when the system state is ON-LINE will initiate the scenario that attempts to bring the equipment OFF-LINE. If the host accepts the off-line request, the equipment will transition to the OFF-LINE state and the new state will be displayed at the bottom of the “Control” group. When the equipment is off-line it will respond to only host messages that request activation of the on-line state.[1] An Sx F0 reply will be made to all other messages.

Pressing **On-Line** when the system state is OFF-LINE will initiate the scenario that attempts to bring the equipment ON-LINE. If the host accepts the on-line request, the equipment will transition to the ON-LINE state and the new state will be displayed at the bottom of the “Control” group. The host can request a transition to the host OFF-LINE state. Once in the host OFF-LINE state, the host can also initiate a request for the equipment to go on-line. If the equipment accepts the host request, the new state will be displayed at the bottom of the “Control” group.

There are two sub-states of the ON-LINE state, REMOTE and LOCAL. When the state is LOCAL, the operator still controls the equipment and the host has the following options and prohibitions:

- The host cannot initiate a process or modify a running process.
- The host cannot modify any EC related to a running process (as specified by the **Process Related:** ☑ option) but can modify ECs while no process is running.
- The host cannot select processes for execution.
- The host can request upload and download of process programs.
- The host can configure alarms, event reporting, and trace data reporting.
- The host can request and receive all data reports.
- The host can perform terminal messages operations.

When the state is REMOTE the host can operate the equipment to the full extent described in this documentation. The equipment can be setup to allow or disallow the following operator actions. The recommended setting is in the second column. See VIDs 9118 to 9131 for the current setting of these restrictions.

Class of Operations	**Recommended in REMOTE Mode?**
Change process-related equipment constants	NO
Change non-process related equipment constants	YES
Initiate process program download	YES
Select Process Program	NO
Start Process Program	NO
Pause/Resume Process Program	NO
Operator Assist	YES

[1] Note that “equipment off-line” does not physically break the communication link with the host. The host can still transmit messages and the equipment will respond with the appropriate message indicating that communication is disabled.

Class of Operations	Recommended in REMOTE Mode?
Material Movement to/from equipment	NA
Equipment specific commands	NA

GEM Requirements

These GEM requirements are specified in Section 4.12.4 of SEMI E30-95:

1. The default control state entered on system initialization can be configured in the GEM Variables database (ECID 9103).
2. The state activated when an attempt to go ON-LINE fails can be configured in the GEM Variables database (ECID 9104).
3. The ON-LINE/OFF-LINE radio-buttons on the GEM Control Panel shown in **Figure 3-3** mimic a momentary switch. These buttons will serve to transition between online and offline.
4. The ON-LINE REMOTE/LOCAL switches are radio buttons on the GEM Control Panel shown in **Figure 3-3**.
5. The control state is displayed in a permanent information area of the GEM Control Panel shown in **Figure 3-3**.
6. The status variable that maintains the status of the control state model is stored in a record in the GEM Variables database (SVID 9003).
7. The "operator command issued" event is stored in a record in the GEM Variables database (SVID 9004).

Chapter 4
SECS-I and HSMS Details

This chapter provides details on the options for establishing the low-level communication channel. It also provides documentation on the SEMI required parameters for the communication channel.

4.1 Introduction

This document describes the AdeptGEM implementation of the SEMI Equipment Communications Standard 1 (SECS-I) and the SEMI Equipment High-Speed SECS Message Service (HSMS). These standards are described in the publication *Book of SEMI Standards 1995: Equipment Automation/Software Volume1* and *Volume2*.

The SECS-I and HSMS layers of the SEMI standards govern the physical link between the equipment and the host. The SECS-I (serial communications) or HSMS (ethernet communication) option runs as an independent task that automatically interacts with the host and the SECS-II layer. Except for setting the relevant communications parameters, you do not have to be concerned with the functional details of SECS-I or HSMS.

The decision to use SECS-I or HSMS is based entirely on the physical medium connecting the equipment and the host. The selection of SECS-I or HSMS is made in equipment constant ID 9300 in the GEM Variables database. To select SECS-I, specify the serial port that will be used for SECS-I communication.

4.2 Additional Documentation

This document details the required options for implementing a GEM compliant piece of equipment. This document will not detail “why” SEMI has decided to perform operations in a given manner, it will detail how AdeptGEM meets those requirements. It will give you the procedures for setting up and starting communication with the host. The SEMI documentation described in section **4.1** is required reading.

The AdeptGEM system is an AIM module and works within the AIM environment. We assume that you are familiar with such tasks as using the mouse, entering data into databases, and starting and stopping the AIM system. If you are not familiar with AIM, please see the ***MotionWare User’s Guide*** or the ***VisionWare User’s Guide*** .

4.3 SECS-I Details

The AdeptGEM SECS-I software is in compliance with the protocols defined in “SEMI E4-91: SEMI Equipment Communications Standard 1: Message Transfer (SECS-I).” Section 7.2.4 of that standard, “Interleaving Messages”, which is optional, is not implemented.

SECS-I Parameters

There are several parameters that can be set to influence the behavior of the AdeptGEM programs. The SECS-I layer parameters are defined in the SEMI specification and are presented here. In some cases, the setting of these parameters will be obvious, based on the physical equipment (the BAUD rate, for example, is limited by the capabilities of the equipment at either end of the communication link). Other parameters will need to be determined based on the behavior of the equipment in the host environment. The Retry and T4 parameters, for example, do not have an intrinsic "correct" value, but depend on the acceptable behavior of the entire system.

All the SECS-I parameters are stored as equipment constants in the GEM Variables database (see **Table 4-1** and **Table 7-2**).

The following table shows the range, default, and resolution for each parameter (T1 - T4 values are in seconds):

Table 4-1
Communication Parameters

Parameter	ECID	Default	Min	Max	Resolution
BAUD	9167	9600	110	38400	see FSET instruction
DEVID	9151	4660	0	32767	1
T1	9106	0.5	0.1	10	0.016
T2	9107	10	0.2	25	0.016
T3	9108	45	1	120	1
T4	9109	45	1	120	1
RTY	9115	3	0	31	1
MS		Defaults to equipment.			

Other SECS-I Requirements

1. Duplicate block detection is optional and is set up with equipment constant ID 9116 in the GEM Variables database. By default, it is enabled.

   Expected delays under normal operating conditions:

Parameter	Definition	Delay
T1	Inter-character time-out	10/Baud rate
T2	Protocol time-out	0.032
T3	Reply time-out	Depends on system configuration (typically 10-30 seconds)
T4	Inter-block time-out	Depends on system configuration (typically 1-2 seconds

2. Multi-block messages are supported for both sending and receiving.
3. The maximum received message size is configurable using ECID 9154 in the GEM Variables database. The absolute maximum size is 64KB (65,535 bytes). (The SECS-II layer may impose limits on the size of many messages.)
4. Message interleaving is supported.
5. Only one device ID is supported per CPU (set with equipment constant ID 9151 in the GEM Variables database).
6. Only one transaction is processed at a time.

4.4 HSMS Details

There are several parameters that can be set to influence the behavior of the AdeptGEM programs. The HSMS layer parameters are defined in the SEMI specification and are presented here. In some cases, the setting of these parameters will be obvious, based on the physical equipment (the IP address, for example, is determined by the setting of the host). Other parameters will need to be determined based on the behavior of the equipment in the host environment. The T7 and T8 parameters, for example, do not have an intrinsic “correct” value, but depend on the acceptable behavior of the entire system.

All the HSMS parameters are stored as equipment constants in the GEM Variables database (see **Table 4-2** and **Table 7-2**).

The following table shows the range, default, and resolution for each parameter (T3 - T8 values are in seconds)

Table 4-2
HSMS Parameter Values

Parameter	ECID	Default	Min	Max	Resolution
T3	9108	45	1	120	1
T5	9110	10	1	240	1
T6	9111	5	1	240	1
T7	9112	10	1	240	1
T8	9113	5	1	120	1
Connect Mode	Auto detected	PASSIVE, ACTIVE			-
Local Entity IP Address and Port number	Configured in AdeptNET	Determined by TCP/IP convention			-
Remote Entity IP Address and Port Number	9300	Determined by TCP/IP convention			-

Table 4-3
HSMS Parameter Description

Parameter	**Description**
T3 Reply Timeout	Specifies maximum time an entity expecting a reply message will wait for that reply.
T5 Connect Separation Timeout	Specifies the time that must elapse between successive attempts to connect to a given remote entity.
T6 Control Transaction Timeout	Specifies the time that a control transaction may remain open before it is considered a communications failure.
T7 NOT SELECTED Timeout	Time that a TCP/IP connection can remain in NOT SELECTED state (i.e., no HSMS activity) before it is considered a communications failure.
T8 Network Intercharacter Timeout	Maximum time between successive bytes of a single HSMS message that may expire before it is considered a communications failure.
Connect Mode	Specifies the logic this local entity will use during HSMS connection establishment.
Local Entity IP Address and Port number	Determines the address on which the local entity will listen for incoming connection requests. (Required for any entity operating in PASSIVE mode.)
Remote Entity IP Address and Port Number	Determines the address of the remote entity to which the local entity will attempt to connect. (Required for any entity operating in ACTIVE mode.)

Other HSMS Requirements

1. The Option Used for Refusing Incoming Connection Requests if the Implementation uses the Passive Mode for TCP/IP Connection Establishment:

   Refuse to listen for or accept the connect request. No action is taken in the local entity, the remote entity’s connect procedure will eventually time out.

2. The Maximum Message Size that can be Received:

   The size of a message received depends on the size of buffers used. The buffer size is configured using equipment constant 9154 in the GEM Variables database.

3. The Maximum Expected Size of Messages Sent:

   The maximum expected size of messages sent will be implementation dependent, but not greater than 64KB.

4. The Maximum Number of Supported Concurrent Open Transactions:

   The maximum number of supported concurrent open transactions is one.

5. The Number of Device IDs Supported and Their Specific Values:

   The number of device IDs supported is one. This ID is configured by equipment constant 9151.

6. Whether or Not the Implementation Supports the Normal or the Restricted Procedure for Terminating Communications:

   The implementation supports the restricted procedure for terminating communications.

7. The Setting of the Host vs. Equipment Parameter:

   In this product, the Adept side is always equipment.

Chapter 5
SECS-II Messages

This chapter provides information about all the SECS-II messages that the AdeptGEM system can exchange with a host. In most cases, the user of AdeptGEM will not need to be concerned with this message detail, because AdeptGEM takes care of correctly formatting messages to the host and decoding messages from the host. However, the host will need to be supplied with this information so it will know the specific message details that are implemented in the AdeptGEM system.

This documentation follows the standards and conventions used by SEMI in documenting standard E5-95. The mnemonics enclosed in less-than/greater-than brackets (for example, "<MDLN>") correspond to the defined data items in SEMI specification E5-95, section 6.5 "Data Item Dictionary."

5.1 General Information

This section provides general information about the AdeptGEM implementation of SECS-II messages.

Manufacturer and Product Number

The baseline AdeptGEM system is developed, supplied, and maintained by Adept Technology, Inc. In addition to the features described in this manual, the system can be enhanced and modified by AIM system customizers, who are responsible for documenting any changes to the baseline system.

The Adept part number for the AdeptGEM system is 90713-01910.

General Description of Equipment Function

The Adept MV controllers deliver unique features and performance in a fully-integrated hardware and software platform. Adept's VME-based controllers provide open architecture to support third-party boards, including PCs, PLCs, I/O and networking cards. Adept controllers offer an integrated solution for motion control and machine vision systems.

Intended Function of the Interface

The AdeptGEM system is intended for host (remote) monitoring and control of an automation system controlled by an Adept controller.

Software Revision Code

To determine the software revision code for the AdeptGEM, watch the "Adept AIM System Initialization" window as the AIM system initializes. The revision code is displayed on a line with the format "GEM Module (3.1D)", which in this case indicates that the AdeptGEM system is version 3.1 edit D.

When AIM is running, the software revision code is stored in the global string variable **$gm.id.** The V$^+$ monitor command LISTS will display the revision code:

```
LISTS $gm.id
```

Changes From Previous Versions

This manual describes the initial version of the AdeptGEM system.

5.2 Supported Messages

This section provides lists of all the SECS-II messages that are understood and initiated by the baseline AdeptGEM system. Additional messages can be added by AIM system customizers. See **Chapter 9** for a description of the **gm.user.strmxx** routines.

Messages Understood by the Equipment

The SECS-II messages listed in **Table 5-1** are understood by the AdeptGEM system. All other SECS-II messages (except SxF0 messages, which are ignored) result in an S9Fx response message.

Table 5-1
SECS-II Messages Understood by the Equipment

Message Received	Response Message Sent
S1F1	S1F2
S1F3	S1F4
S1F11	S1F12
S1F13	S1F14
S1F15	S1F16
S1F17	S1F18
S2F13	S2F14
S2F15	S2F16
S2F17	S2F18
S2F23	S2F24
S2F29	S2F30
S2F31	S2F32
S2F33	S2F34
S2F35	S2F36
S2F37	S2F38

Table 5-1
SECS-II Messages Understood by the Equipment (Continued)

Message Received	Response Message Sent
S2F39	S2F40
S2F41	S2F42
S2F45	S2F46
S2F47	S2F48
S5F3	S5F4
S5F5	S5F6
S5F7	S5F8
S6F15	S6F16
S6F17	S6F18
S6F19	S6F20
S6F21	S6F22
S7F1	S7F2
S7F3	S7F4
S7F5	S7F6
S7F17	S7F18
S7F19	S7F20
S10F3	S10F4
S10F5	S10F6
S10F9	S10F10

Messages Initiated by the Equipment

The SECS-II messages listed in **Table 5-2** are initiated by the AdeptGEM system.

Table 5-2
SECS-II Messages Initiated by the Equipment

Message Sent	Expected Response Message
S1F1	S1F2
S1F13	S1F14
S2F17	S2F18
S5F1	S5F2
S6F1	S6F2
S6F5	S6F6
S6F11	S6F12
S7F1	S7F2
S7F3	S7F4
S9F1	-
S9F3	-
S9F5	-
S9F6	-
S9F7	-
S9F13	-
S10F1	S10F2

5.3 Message Details

This section details the message streams and functions that are supported by the Adept SECS-II implementation. (Note, however, that system customizers can add support for additional host messages, and they can modify the response messages from the equipment.)

Stream 1: Equipment Status — This stream provides a means for exchanging information about the status of the equipment.

S1, F0 Abort Transaction (S1F0) *S, H <-> E*

Description: Used in lieu of an expected reply to abort a transaction. Function 0 is defined in every stream and has the same meaning in every stream.

Structure: Header only.

S1, F1 Are You There Request (R) *S, H <-> E, reply*

Description: Establishes if the equipment is on line. A function 0 response to this message means the communication is inoperative. In the equipment, a function 0 is equivalent to a time-out on the receive timer after issuing S1,F1 to the host.

Structure: Header only.

S1, F2 On Line Data (D) *S, H <-> E*

Description: Data signifying the equipment is on line.

Structure: L, 2

1. <MDLN> (ASCII data) "A" + 4 bytes from V$^+$ function ID(1,1)
2. <SOFTREV> Software version ID(3,1) and revision ID(4,1) (ASCII data, 4 bytes), or value from equipment constant 9301 (see ECID 9102)

Exceptions: The host sends a zero-length list to the equipment.

S1, F3 Select Equipment Status Request (SSR) *S, H -> E, reply*

Description: This is a request to the equipment to report selected values of its status.

Structure: L, n

1. <SVID1> (unsigned 1, 2, 4, or 8 byte integer data; 1 – 32767)

.

.

n. <SVIDn>

The obsolete structure <SVID1,..., SVIDn> is also supported.

See **Table 7-2** for the predefined SVIDs.

Exceptions: A zero-length list means report all SVIDs.

S1, F4 Formatted Status Data (FSD) *M, H <- E*

Description: The equipment reports the value of each SVID requested by S1,F3. The host remembers the names of the variables requested.

Structure: L, n

1. <SV1> (any allowed format)

.
.

n. <SVn>

Exceptions: If n = 0, no response can be made. A zero length returned for SVi means that SVIDi does not exist.

S1, F11 Status Variable Namelist Request (SVNR) *S, H -> E, reply*

Description: A request to the equipment to identify certain status variables.

Structure: L, n

1. <SVID1> (unsigned 1, 2, 4, or 8 byte integer data; 1 – 32767)

.
.

n. <SVIDn>

Exceptions: A zero-length list means report all SVIDs.

S1, F12 Status Variable Namelist reply (SVNRR) *M, H<- E*

Description: The equipment reports to the host the name and units of the requested SVs.

Structure: L, n

1. L, 3
   1. <SVID1> (2 byte unsigned integer; 1 – 32767)
   2. <SVNAME1> (ASCII; 0 – 15 characters)
   3. <UNITS1> (ASCII; 0 – 20 characters)

.
.

n. L, 3
   1. <SVID1>
   2. <SVNAME1>
   3. <UNITS1>

S1, F13 Establish Communications Request (CR) *S, H <-> E, reply*

Description: This message provides a means of initializing communications at start-up or after a communications break.

Structure: L, 2

1. <MDLN> (ASCII data) "A" + 4 bytes from V^+ function ID(1,1)
2. <SOFTREV> Software version ID(3,1) and revision ID(4,1) (ASCII data, 4 bytes), or value from equipment constant 9301 (see ECID 9102)

Exceptions: Host sends a zero-length list.

S1, F14 Establish Communications Request Acknowledge (CRA) *S, H <-> E*

Description: Accept or deny Establish Communications Request.

Structure: L, 2

1. <COMMACK> (1 byte: 0 = accepted, 1 = denied)
2. L, 2
   1. <MDLN> (ASCII data) "A" + 4 bytes from V^+ function ID(1,1)
   2. <SOFTREV> Software version ID(3,1) and revision ID(4,1) (ASCII data, 4 bytes), or values from Equipment Constant ID 9301

Exceptions: The host sends a zero-length list for item 2.

S1, F15 Request OFF-LINE (ROFL) *S, H -> E, reply*

Description: The host requests that the equipment transition to the off-line state.

Structure: Header only.

S1, F16 OFF-LINE Acknowledge (OFLA) *S, H <- E*

Description: Acknowledge or error.

Structure: <OFLACK> (1 byte, always 0)

S1, F17 Request ON-LINE(RONL) *S, H -> E, reply*

Description: The host requests that the equipment transition to the on-line state.

Description: Header only.

S1, F18 ON-LINE Acknowledge (ONLA) *S, H <- E*

Description: Acknowledge or error.

Structure: <ONLACK> (1 byte: 0 = accepted, 1 = not allowed, 2 = already on-line)

Stream 2: Equipment Control and Diagnostics — Messages that deal with control of the equipment from the host. This includes all remote operations and equipment self-diagnostics and calibration, but specifically excludes the control operations that are associated with material transfer (stream 4), loading of executive and boot programs (stream 8), and all file and operating system calls (streams 7 and 13).

S2, F0 Abort Transaction (S2F0) *S, H <-> E*

Description: Used in lieu of an expected reply to abort a transaction. Function 0 is defined in every stream and has the same meaning in every stream.

S2, F13 Equipment Constant Request (ECR) *S, H -> E, reply*

Description: This is a request to the equipment to report the values of selected equipment constants.

Structure: L, n

1. <ECID1> (unsigned 1, 2, 4, or 8 byte integer data; 1 – 32767)

.

.

n. <ECIDn>

The obsolete structure <ECID1,..., ECIDn> is also supported.

Exceptions: A zero-length list or item means report all ECVs according to a predefined order.

S2, F14 Equipment Constant Data (ECD) *M, H <- E*

Description: Data response to S2,F13 in the order requested.

Structure: L, n

1. <ECV1> (any allowed data type)

.

.

n. <ECVn>

S2, F15 New Equipment Constant Send (ECS) *S, H -> E, reply*

Description: Change one or more equipment constants.

Structure: L, n

1. L, 2
   1. <ECID1> (unsigned 1, 2, 4, or 8 byte integer data; 1 – 32767)
   2. <ECV1> (any allowed data type)

.

.

n. L, 2
   1. <ECIDn>
   2. <ECVn>

S2, F16 New Equipment Constant Acknowledge (ECA) *S, H <- E*

Description: Acknowledge or error. If any non-zero value is returned, no ECIDs will be changed.

Structure: <EAC> (1 byte binary: 0 = OK, 1 = denied—at least one constant does not exist, 2 = denied—busy, 3 = denied—at least one constant is out of range or conversion was not successful, 4 = denied—at least one constant cannot be changed by the host)

S2, F17 Date and Time Request (DTR) *S, H <-> E, reply*

Description: Request the time as maintained by the equipment.

Structure: Header only.

S2, F18 Data and Time Data (DTD) *S, H <-> E*

Description: Actual time data.

Structure: <TIME> (12 character ASCII in “yymmddhhmmss” format)

Exceptions: A zero-length item means the time is undefined.

S2, F23 Trace Initialization Send (TIS) S, H -> E, reply

Description: This function provides a way to sample a subset of status variables as a function of time.

Structure: L, 5

1. <TRID> (unsigned 1, 2, 4, or 8 byte integer data; 0 – 65535)
2. <DSPER> (6 byte ASCII in format “hhmmss”)
3. <TOTSMP> (unsigned 1, 2, 4, or 8 byte integer data; 0 – 65535)
4. <REPGSZ> (unsigned 1, 2, 4, or 8 byte integer data; 1 – 65535)
5. L, n
   1. <SVID1> (unsigned 1, 2, 4, or 8 byte integer data; 1 – 32767)
   .
   .
   n. <SVIDn>

The following obsolete structure is also supported:

1. <TRID> (unsigned 1, 2, 4, or 8 byte integer data)
2. <DSPER> (6 byte ASCII in format “hhmmss”)
3. <TOTSMP> (unsigned 1, 2, 4, or 8 byte integer data)
4. <REPGSZ> (unsigned 1, 2, 4, or 8 byte integer data)
5. <SVID1>,. . . .,<SVIDn>

S2, F24 Trace Initialization Acknowledge (TIA) S, H <- E

Description: Acknowledge or error.

Structure: <TIAACK> (1 byte binary: 1 = OK, 1 = too many SVIDs, 2 = no more traces allowed, 3 = invalid period)

S2, F29 Equipment Constant Namelist Request (ECNR) S, H -> E, reply

Description: This function allows the host to retrieve basic information about equipment constants that are available in the equipment.

Structure: L, n

1. <ECID1> (unsigned 1, 2, 4, or 8 byte integer data; 1 – 32767)
.
.
n. <ECIDn>

Exceptions: A zero-length list means send information for all ECIDs.

S2, F30 Equipment Constant Namelist (ECN) *M, H <- E*

Description: Data response to S2,F29.

Structure: L, n

1. L, 6
    1. <ECID1> (2 byte unsigned integer; 1 – 32767)
    2. <ECNAME1> (ASCII; 0 – 15 characters)
    3. <ECMIN1> (any allowed data type)
    4. <ECMAX1> (any allowed data type)
    5. <ECDEF1> (any allowed data type)
    6. <UNITS1> (ASCII; 0 – 20 characters)

.
.

n. L, 6
1. <ECIDn>
2. <ECNAMEn>
3. <ECMINn>
4. <ECMAXn>
5. <ECDEFn>
6. <UNITSn>

S2, F31 Date And Time Set Request (DTS) *S, H -> E, reply*

Description: Set the equipment time base.

Structure: <TIME> (ASCII, 12 characters in the form “yymmddhhmmss”)

S2, F32 Date And Time Set Acknowledge (DTA) *S, H <- E*

Description: Acknowledge receipt of the time and date.

Structure: <TIACK> (1 byte binary: 0 = OK, 1 = error)

S2, F33 Define Report (DR) *M, H -> E, reply*

Description: The message allows the host to define a group of reports for the equipment. If this is a multi-block message, it must be preceded by the S2,F39/S2,F40 Inquire/Grant transaction.

Structure: L, 2

1. <DATAID> (ignored, unsigned 1, 2, 4, or 8 byte integer data)
2. L, a
    1. L, 2
        1. <RPTID1> (unsigned 1, 2, 4, or 8 byte integer data; 1 – 32767)
        2. L, b
            1. <VID> (unsigned 1, 2, 4, or 8 byte integer data; 1 – 32767)

```
            .
            .
            b.  <VIDb>
    .
    .
    a.  L, 2
        1.  <RPTIDa>
        2.  L, c
            1.  <VIDl>
            .
            .
            c.  <VIDc>
```

Exceptions: A zero-length list following <DATAID> deletes all report definitions and associated links. A zero-length list following <RPTID> deletes that report, and all <CEID> links to that report.

S2, F34 Define Report Acknowledge (DRA) *S, H <- E*

Description: Acknowledge or error. If an error occurs, the entire message is rejected; i.e.,partial changes are not allowed.

Structure: <DRACK> (1 byte binary: 0 = accept, 1 = insufficient memory space or other serious error, 2 = invalid format, 3 = at least one <RPTID> already defined, 4 = at least one <VID> does not exist.

S2, F35 Link Event Report (LER) *M, H -> E, reply*

Description: This message allows the host to link reports to an event <CEID>. These linked events will default to disabled upon linking. If this is multi-block, it most be preceded by the S2,F39/S2,F40 Inquire/Grant transaction.

Structure: L, 2

```
1.  <DATAID> (ignored, unsigned 1, 2, 4, or 8 byte integer data)
2.  L, a
    1.  L, 2
        1.  <CEID1> (unsigned 1, 2, 4, or 8 byte integer data; 1 – 32767)
        2.  L, b
            1.  <RPTID1> (unsigned 1, 2, 4, or 8 byte integer data; 1 – 32767)
            .
            .
            b.  <RPTIDb>
    .
    .
    a.  L, 2
        1.  <CEIDa>
        2.  L, c
            1.  <RPTID1>
            .
            .
```

c. <RPTIDc>

Exceptions: A list of zero length following <CEID> deletes all report links to that event.

S2, F36 Link Event Report Acknowledge (LERA) S, H <- E

Description: Acknowledge or error. If an error condition is detected the entire message is rejected, partial changes are not allowed.

Structure: <LRACK> (1 byte binary: 0 = OK, 1 = out of memory, 2 = invalid format, 3 = at least one <CEID> is already defined, 4 = at least one <CEID> does not exist, 5 = at least one <RPTID> does not exist.

S2, F37 Enable/Disable Event Report (EDER) S, H -> E, reply

Description: Allows the host to enable/disable reporting for a group of events <CEIDs>.

Structure: L, 2

1. <CEED> (1 byte Boolean: TRUE = enabled, FALSE = disabled)
2. L, n
   1. <CEID1> (unsigned 1, 2, 4, or 8 byte integer data; 1 – 32767)
   .
   .
   n. <CEIDn>

Exceptions: A zero-length list means all <CEIDs>.

S2, F38 Enable/Disable Event Report Acknowledge (EERA) S, H <- E

Description: Acknowledge or error. If an error condition is detected the entire message is rejected, partial changes are not allowed.

Structure: <ERACK> (1 byte binary: 0 = accepted, 1 = denied because at least one <CEID> does not exist)

S2, F39 Multi-block Inquire (DMBI) S, H -> E, reply

Description: Request permission to transmit multi-block S2,F33; S2,F35; or S2,F45 message.

Structure: L, 2

1. <DATAID> (unsigned 1, 2, 4, or 8 byte integer data)
2. <DATALENGTH> (unsigned 1, 2, 4, or 8 byte integer data; 255 – 65,535)

NOTE: Messages are limited to 64KB (or less) according to ECID 9154.

S2, F40 Multi-block Grant (DMBG) *S, H <- E*

Description: Grant permission to send multi-block message.

Structure: <GRANT> (1 byte binary: 0 = permission granted, 2 = buffer size not large enough to handle message)

S2, F41 Host Command Send (HCS) *S, H -> E, reply*

Description: Host requests the Equipment perform specified remote command with the associated parameters. See **section 3.9 on page 33** for information on how the equipment interprets remote commands.

Structure: L, 2

1. <RCMD> (command name, ASCII data)
2. L, n (number of parameters)
    1. L, 2
        1. <CPNAME1> (ASCII data)
        2. <CPVAL1> (ASCII data)

    .
    .

    n. L, 2
        1. <CPNAMEn> (ASCII data)
        2. <CPVALn> (any allowed data type)

S2, F42 Host Command Acknowledge (HCA) *S, H <- E*

Description: Acknowledge or error response to host command request.

If the host command request causes an AIM control sequence to be executed (see **section 3.9 on page 33**), the response code in this message will be 4. However, no event will be generated automatically at the completion of the control sequence—only at the completion of any non-control sequence that the control sequence initiates.

Structure: L, 2

1. <HCACK> (1 byte binary data: 0 = OK, 1 = command does not exist, 2 = cannot perform command now, 3 = at least one parameter is invalid, 4 = command will be performed with completion signaled later by an event.)
2. L, n
    1. L, 2
        1. <CPNAME1> (ASCII data)
        2. <CPACK1> (1 byte binary)

    .
    .

    n. L, 2
        1. <CPNAMEn>
        2. <CPACKn>

S2, F45 Define Variable Limit Attributes (DVLA) *M, H -> E, reply*

Structure: L, 2

```
1.  <DATAID> (unsigned 1, 2, 4, or 8 byte integer data)
2.  L, m
    1.  L, 2
        1.   <VID1> (unsigned 1, 2, 4, or 8 byte integer data; 1 – 32767)
        2.  L, n
            1.  L, 2
                1.  <LIMITID1> (1 binary byte; 1 – 7)
                2.  L, p (p = 0 or 2)
                    1.  <UPPERDB1> (any allowed data type)
                    2.  <LOWERDB1> (any allowed data type)
            .
            .
            n.  L, 2
                1.  <LIMITIDn>
                2.  L, p (p = 0 or 2)
                    1.  <UPPERDBn>
                    2.  <LOWERDBn>
    .
    .
    m.  L, 2
        1.  <VIDm>
        2.  L, n
            1.  L, 2
                1.  <LIMITID1>
                2.  L, p (p = 0 or 2)
                    1.  <UPPERDB1>
                    2.  <LOWERDB1>
            .
            .
            n.  L, 2
                1.  <LIMITIDn>
                2.  L, p (p = 0 or 2)
                    1.  <UPPERDBn>
                    2.  <LOWERDBn>
```

Exceptions: A zero-length list, m = 0, sets all limit values for all monitored VIDs to “undefined”. A zero-length list, n = 0, sets all limits values for the VID to “undefined”. A zero-length list, p = 0, sets that limit to “undefined”.

S2, F46 Variable Limit Attribute Acknowledge (VLAA) *M, H <- E*

Description: Acknowledge definition of variable limit attributes or report error.

Structure: L, 2

1. <VLAACK> (1 binary byte: 0 = OK, 1 = error in defining limit attribute, 2 = cannot perform at this time)
2. L, m
    1. L, 3
        1. <VID1> (unsigned 1, 2, 4, or 8 byte integer data; 1 – 32767)
        2. <LVACK1> (1 binary byte: 1 = variable does not exist, 2 = this variable cannot be monitored, 3 = this variable has already been used in the same message, 4 = limit value error)
        3. L, n (n = 0 or 2)
            1. <LIMITID> (1 binary byte; 1 – 7)
            2. <LIMITACK> (1 binary byte: 1 = <LIMITID> does not exist or is out of legal range of 1 – 7, 2 = UPPERDB > LIMITMAX, 3 = LOWERDB < LIMITMIN, 4 = UPPERDB < LOWERDB, 5 = illegal format specified for UPPERDB or LOWERDB, 6 = ASCII value can not be translated to numeric, 7 = duplicate limit definition for this variable.

    .

    .

    m. L, 3
        1. <VIDm>
        2. <LVACKm>
        3. L, n (n = 0 or 2)
            1. <LIMITID>
            2. <LIMITACK>

Exceptions: A zero-length list, m = 0, indicates no valid limit attributes. A zero-length list, n = 0, indicates no invalid limit values for that <VID>.

S2, F47 Variable Limit Attribute Request (VLAR) *S, H -> E, reply*

Description: Allows the host to query the equipment for current variable limit attribute definitions.

Structure: L, n

1. <VID1> (unsigned 1, 2, 4, or 8 byte integer data; 1 – 32767)

.

.

n. <VIDn>

Exceptions: A zero-length list, n = 0, requests a list of all <VID> values that can have a variable limit structure.

S2, F48 Variable Limit Attributes Send (VLAS) *M, H <- E*

Description: Equipment sends values and variable limit attribute definitions in the order requested.

Structure: L, m

```
1.  L, 2
    1.  <VID1> (unsigned 1, 2, 4, or 8 byte integer data; 1 – 32767)
    2.  L, p (p = 0 or 4)
        1.  <UNITS1> (ASCII data; 0 – 20 characters)
        2.  <LIMITMIN1> (minimum value allowed for variable)
        3.  <LIMITMAX1> (maximum value allowed for variable)
        4.  L, n
            1.  L, 3
                1.  <LIMITID1> (1 byte binary data; 1 – 7)
                2.  <UPPERDB1> (any allowed data type)
                3.  <LOWERDB1> (any allowed data type)
            .
            .
            n.  L, 3
                1.  <LIMITIDn>
                2.  <UPPERDBn>
                3.  <LOWERDBn>
.
.
m.  L, 2
    1.  <VIDm>
    2.  L, p
        1.  <UNITSm>
        2.  <LIMITMINm>
        3.  <LIMITMAXm>
        4.  L, n
            1.  L,3
                1.  <LIMITID1>
                2.  <UPPERDB1>
                3.  <LOWERDB1>
            .
            .
            n.  L, 3
                1.  <LIMITIDn>
                2.  <UPPERDBn>
                3.  <LOWERDBn>
```

Exceptions: A zero-length list, p = 0, indicates that limits are not supported for the <VID>. A zero-length list, n = 0, indicates no limits are currently defined for the specified variable.

Stream 5: Exception Reporting — This stream contains messages regarding digital and analog equipment alarms. The alarms are generated by the equipment in response to changing conditions detected by the equipment.

S5, F0 Abort Transaction (S5F1) *S, H <-> E*

Description: Used in lieu of an expected reply to abort a transaction. Function 0 is defined in every stream and has the same meaning in every stream.

S5, F1 Alarm Report Send (ARS) *S, H <- E, reply*

Description: Reports a change in or presence of an alarm condition. One message will be issued when the alarm is set, and one message will be issued when the alarm is cleared. Irrecoverable errors and attention flags may not have a corresponding clear message.

Structure: L, 3

1. <ALCD> (1 byte binary data)
   bit 8 indicates the state of the alarm: bit set (= 1) means alarm is set, bit clear (= 0) means alarm is clear.
   bits 7 – 1 alarm category (not used by the GEM standard, but supported by the AdeptGEM system).
2. <ALID> (2 byte unsigned integer; 1 – 32767)
3. <ALTX> (ASCII data; 0 – 40 characters)

S5, F2 Alarm Report Acknowledge (ARA) *S, H -> E*

Description: Acknowledge or error

Structure: <ACKC5> (1 byte binary: 0 = OK, 1 = error)

S5, F3 Enable/Disable Alarm Send (EAS) *S, H -> E, [reply]*

Description: This message will change the state of the enable bit in the equipment. The enable bit determines if the alarm will be sent to the host.

Structure: L, 2

1. <ALED> (1 byte binary: 0 – 127 = disable alarm, 128 – 255 = enable alarm)
2. <ALID> (any allowed data type)

Exceptions: A zero length for <ALID> means all alarms.

S5, F4 Enable/Disable Alarm Acknowledge (EAA) *S, H <- E*

Description: Acknowledge or error.

Structure: <ACKC5> (1 byte binary: 0 = OK, 1 = error)

S5, F5 List Alarms Request (LAAR) *S, H -> E, reply*

Description: Requests the equipment to send binary and analog alarm information.

Structure: <ALID1,..., ALIDn> (unsigned 1, 2, 4, or 8 byte integer data)

Exceptions: A zero-length item means send all possible alarms.

S5, F6 Alarm List Data (LAD) *M, H <- E*

Description: Return the alarm data known to the equipment.

Structure: L, n

1. L, 3
   1. <ALCD1> (1 byte binary: bit 8 = 1 if alarm is set, bits 7 - 1 = alarm category)
   2. <ALID1> (2 byte unsigned integer; 1 – 32767)
   3. <ALTX1> (ASCII data; 0 – 40 characters)

.
.

n. L, 3
   1. <ALCDn>
   2. <ALIDn>
   3. <ALTXn>

Exceptions: If n = 0, no response can be made. A zero-length item returned for <ALCDi> or <ALTXi> means that the value does not exist.

S5, F7 List Enabled Alarm Request (LEAR) *S, H ->E, reply*

Description: Host requests list of enabled alarms.

Structure: Header only.

S5, F8 List Enabled Alarm Data (LEAD) *M, H <- E*

Description: Equipment returns list of all enabled alarms.

Structure: Same as S5,F6.

Stream 6: Data Collection —This stream is intended to cover the needs of in-process measurements and equipment monitoring.

S6, F0 Abort Transaction (S6F10) *S, H <-> E*

Description: Used in lieu of an expected reply to abort a transaction. Function 0 is defined in every stream and has the same meaning in every stream.

S6, F1 Trace Data Send (TDS) *S, H <- E, reply*

Description: Equipment returns samples to the host according to the trace setup done by S2,F23.

Structure: L, 4

1. <TRID> (2 byte unsigned integer; 0 – 65535)
2. <SMPLN> (2 byte unsigned integer; 1 – 65535)
3. <STIME> (ASCII data in the form "yymmddhhmmss")
4. L, n
   1. <SV1> (any allowed data type)
   .
   .
   n. <SVn>

Exceptions: A zero-length <STIME> means no value is given and that the time is to be derived from <SMPLN> along with knowledge of the request.

S6, F2 Trace Data Acknowledge (TDA) *S, H -> E*

Description: Acknowledge or error

Structure: <ACKC6> (1 byte binary: 0 = OK, 1 = error)

S6, F5 Multi-block Data Send Inquire (MBI) *S, H <- E, reply*

Description: If an S6,F3/F9/F11/F13 message can be multi-block, this request must be made before transmission.

Structure: L, 2

1. <DATAID> (2 byte unsigned integer; normally 0)
2. <DATALENGTH> (2 byte unsigned integer; 0 – 65535)

S6, F6 Multi-block Grant (MBG) *S, H ->E*

Description: Allow multi-block transmission.

Structure: <GRANT6> (1 byte binary: 0 = OK, 1 = busy, 2 = not interested, >2 = other error)

S6, F11 Event Report Send (ERS) — *M, H <- E, reply*

Description: Equipment sends a defined, linked, and enabled group of reports to the host upon the occurrence of an event <CEID>. Multi-block transmissions must be preceded by S6,F5/S6,F6 grant.

Structure: L, 3

```
1.  <DATAID> (2 byte unsigned integer; normally 0)
2.  <CEID> (2 byte unsigned integer; 1 – 32767)
3.  L, a
    1.  L, 2
        1.  <RPTID1> (2 byte unsigned integer; 1 – 32767)
        2.  L, b
            1.  <V1> (any allowed data type)
            .
            .
            b.  <Vb>
    .
    .
    a.  L, 2
        1.  <RPTIDa>
        2.  L, c
            1.  <V1>
            .
            .
            c.  <Vc>
```

Exceptions: If there are no reports linked to the event a “null” report is assumed. A zero-length list, a = 0, means there are no reports linked to the given <CEID>.

S6, F12 Event Report Acknowledge (ERA) — *S, H -> E*

Description: Acknowledge or error.

Structure: <ACKC6> (1 byte binary: 0 = OK, > 0 = error)

S6, F15 Event Report Request (ERR) — *S, H -> E, reply*

Description: The host requests a given report group from the equipment.

Structure: <CEID> (unsigned 1, 2, 4, or 8 byte integer data; 1 – 32767)

S6, F16 Event Report Data (ERD) — *M, H <- E*

Description: Equipment sends reports linked to given <DEID> to host.

Structure: L, 3

```
1.  <DATAID> (2 byte unsigned integer; normally 0)
2.  <CEID> (2 byte unsigned integer; 1 – 32767)
3.  L, a
    1.  L, 2
        1.  <RPTID1> (2 byte unsigned integer; 1 – 32767)
        2.  L, b
            1.  <V1> (any allowed data type)
            .
            .
            b.  <Vb>
    .
    .
    a.  L, 2
        1.  <RPTIDa>
        2.  L, c
            1.  <V1>
            .
            .
            c.  <Vc>
```

Exceptions: A zero-length list, a = 0, means there are no reports linked to the given <CEID>.

S6, F17 Annotated Event Report Request (AERR) S, H -> E, reply

Description: Same as S6,F15, buts requests annotated reports.

Structure: <CEID>

S6, F18 Annotated Event Report Data (AERD) M, H <- E

Description: Equipment sends annotated reports linked to a given <CEID>.

Structure: L, 3

```
1.  <DATAID> (2 byte unsigned integer; normally 0)
2.  <CEID> (2 byte unsigned integer; 1 – 32767)
3.  L, a
    1.  L, 2
        1.  <RPTID1> (2 byte unsigned integer; 1 - 32767)
        2.  L, b
            1.  L, 2
                1.  <VID1> (2 byte unsigned integer; 1 - 32767)
                2.  <V1> (any allowed data type)
            .
            .
            b.  L, 2
                1.  <VIDb>
                2.  <Vb>
    .
    .
    a.  L, 2
        1.  <RPTIDa>
        2.  L, c
            1.  L, 2
                1.  <VID1>
                2.  <V1>
            .
            .
            c.  L, 2
                1.  <VIDc>
                2.  <Vc>
```

Exceptions: A zero-length list, a = 0, means there are no reports linked to the given <CEID>.

S6, F19 Individual Report Request (IRR) — *S, H -> E, reply*

Description: The host request a defined report from the equipment.

Structure: <RPTID> (unsigned 1, 2, 4, or 8 byte integer data; 1 – 32767)

S6, F20 Event Report Data (IRD) — *M, H <- E*

Description: Equipment sends variable data defined for the given <RPTID> to the host.

Structure: L, n

```
1.  <V1> (any allowed data type)
.
.
n.  <Vn>
```

Exceptions: A zero-length list means <RPTID> is not defined.

S6, F21 Annotated Individual Report Request (AIRR) — *S, H -> E, reply*

Description: Host requests an annotated defined report from the equipment.

Structure: <RPTID> (unsigned 1, 2, 4, or 8 byte integer data; 1 - 32767)

S6, F22 Annotated Individual Report Data (AIRD) — *M, H <- E*

Description: Equipment returns requested annotated variable data defined for the given <RPTID> to the host.

Structure: L, n

```
1.  L, 2
```

1. <VID1> (2 byte unsigned integer; 1 - 32767)
2. <V1> (any allowed data type)

.

.

n. L, 2

1. <VIDn>
2. <Vn>

Exceptions: A zero-length list, n = 0, indicates that <RPTID> is not defined.

Stream 7: Process Program Management — The functions in this stream are used to manage and transfer process control programs.

S7, F0 Abort Transaction (S7F0) *S, H <-> E*

Description: Used in lieu of an expected reply to abort a transaction. Function 0 is defined in every stream and has the same meaning in every stream.

S7, F1 Process Program Load Inquire (PPI) *S, H <-> E, reply*

Description: This message is used to initiate the transfer of a process program or disk file. It must be called prior to S7F3/S7F4. See **section 3.11 on page 38** for information on how the equipment interprets <PPID>.

Structure: L,2

1. <PPID> (ASCII data, 1 – 15 characters)
2. <LENGTH> (unsigned 1, 2, 4, or 8 byte integer data)

S7, F2 Process Program Load Grant (PPG) *S, H <-> E*

Description: This message gives permission for the process program to be loaded.

Structure: <PPGNT> (1 byte binary: 0 = OK, 1 = program already exists, 2 = not enough space available, 3 = invalid <PPID>, 4 = busy – try later, 5 = program cannot be accepted)

S7, F3 Process Program Send (PPS) *S, H <-> E, reply*

Description: The program or disk file initiated by S7F1/S7F2 is transferred.

Structure: L,2

1. <PPID> (ASCII data; 1 – 15 characters)
2. <PPBODY> (ASCII data)

S7, F4 Process Program Acknowledge *S, H <- E*

Description: Acknowledge or error

Structure: <ACKC7> (1 byte binary: 0 = accepted, 1 = permission not granted, 2 = length error, 5 = other error [e.g., validation failed])

S7, F5 Process Program Request (PPR) *S, H -> E, reply*

Description: This message is used to request the transfer of a process program or disk file from the equipment to the host. See **section 3.11 on page 38** for information on how the equipment interprets <PPID>.

Structure: <PPID> (ASCII data; 1 – 15 characters)

S7, F6 Process Program Data (PPD) *M, H <- E*

Description: This message is used to transfer the program or file requested by S7,F5.

Structure: L, 2

1. <PPID> (ASCII data; 1 – 15 characters)
2. <PPBODY> (ASCII data)

Exceptions: A zero-length list means request denied.

S7, F17 Delete Program Send *S, H -> E, reply*

Description: This message is used by the host to request the equipment to delete a process file from disk. See **section 3.11 on page 38** for information on how the equipment interprets <PPID>.

Structure: L, n (number of programs to be deleted)

1. <PPID1> (ASCII data; 1 – 15 characters)

.
.

n. <PPIDn>

Exceptions: A zero-length list means delete all the process programs. (In that case, the AdeptGEM system will delete all the resource modules in the directory specified by ECID 9302.)

S7, F18 Delete Process Program Acknowledge (DPA) *S, H <- E*

Description: Acknowledge or error

Structure: <ACKC7> (1 byte binary: 0 = accepted, 1 = permission not granted, 2 = length error, 4 = <PPID> not found)

S7, F19 Current EPPD Request (RER) *S, H -> E, reply*

Description: This message is used to request the current directory of disk files in the directory specified by ECID 9302.

Structure: None, header only

S7, F20 Transmit Program Directory *M, H <- E*

Description: This message is used to transmit the EPPD (equipment process program directory). See **section 3.11 on page 38** for information on how the equipment interprets the EPPD.

Structure: L, n (number of process files in the directory specified by ECID 9302)

1. <PPID1> (ASCII data; 1 – 15 characters)

.

.

n. <PPIDn>

Stream 9: System Errors — This stream provides a method of informing the host that a message block has been received that cannot be handled or that a time-out on a receive timer has occurred.

S9, F0 Abort Transaction (S9F0) *S, H <-> E*

Description: Used in lieu of an expected reply to abort a transaction. Function 0 is defined in every stream and has the same meaning in every stream.

S9, F1 Unrecognized Device ID (UDN) *S, H <- E*

Description: The device ID in the message block header did not correspond to any known device ID in the node detecting the error.

Structure: Return the header block of the message, <MHEAD>

S9, F3 Unrecognized Stream Type (USN) *S, H <- E*

Description: The equipment does not recognize the stream type in the message block header.

Structure: Return the header block of the message, <MHEAD>

S9, F5 Unrecognized Function Type (UFN) *S, H <- E*

Description: The equipment does not recognize the function type in the message block header.

Structure: Return the header block of the message, <MHEAD>

S9, F7 Illegal Data (IDN) *S, H <- E*

Description: This message indicates that the stream and function were recognized but the associated data format could not be interpreted.

Structure: Return the header block of the message, <MHEAD>

S9, F9 Transaction Timer Time-out (TTN) *S, H <- E*

Description: This message indicates that a transaction (receive) timer has timed out and that the corresponding transaction has been aborted. It is up to the host to respond to this error in an appropriate manner to keep the system operational.

Structure: Stored header related to the transaction timer, <SHEAD>

S9, F11 Data Too Long (DLN) *S, H <- E*

Description: Used to indicate that the equipment has been sent more data than it can handle.

Structure: Return the header block of the message, <MHEAD>

S9, F13 Inter Block Time-out (TTN) *S, H <- E*

Description: Conversation timeout expired.

Structure: <MEXP> (message expected, in the form SxxFyy; ASCII)
<EDID> (if S7F3 is returned: previously referenced <PPID>; ASCII)
(if S2F33 is returned: previously referenced <DATAID>; integer)

Stream 10: Terminal Services — The functions of this stream are to pass textual messages between operator terminals attached to the host or equipment.

S10, F0 Abort Transaction *S, H <-> E*

Description: Used in lieu of an expected reply to abort a transaction. Function 0 is defined in every stream and has the same meaning in every stream.

S10, F1 Terminal Request (TRN) *S, H <- E, [reply]*

Description: Send a terminal text message to the host.

Structure: L, 2

1. <TID> (1 byte binary; value 0)
2. <TEXT> (message text; ASCII data)

S10, F2 Terminal Request Acknowledge (TRA) *S, H -> E*

Description: Acknowledge or error.

Structure: <ACKC10> (1 byte binary: 0 = accepted for display, 1 = message will not be displayed, 2 = terminal not available)

S10, F3 Terminal Display, Single (VTN) *S, H -> E, reply*

Description: Data to be displayed on the equipment terminal.

Structure: L, 2

1. <TID> (ignored, 1 byte binary)
2. <TEXT> (ASCII data; 0 – 238 characters)

S10, F4 Terminal Display, Single Acknowledge (VTA) *S, H <- E*

Description: Acknowledge or error

Structure: <ACKC10> (1 byte binary: 0 = accepted for display, 1 = message will not be displayed)

S10, F5 Terminal Display, Multi-Block (VTN) *M, H -> E, [reply]*

Description: Host sends data to be displayed on the equipment terminal.

Structure: L, 2

1. <TID> (ignored, 1 byte binary)
2. L, n
   1. <TEXT1> (ASCII data, 238 characters maximum)
   .
   .
   n. <TEXTn>

S10, F6 Terminal Display, Multi-Block Acknowledge (VMA) *S, H <- E*

Description: Acknowledge or error.

Structure: <ACKC10> (1 byte binary: 0 = accepted for display, 1 = message will not be displayed)

S10, F9 Broadcast (BCN) *S, H -> E, [reply]*

Description: This function is generally the same as S10,F3 except that specific TID in each equipment need not be specified. Instead, the text is directed to each terminal in the equipment when the function is received. This function assumes that this feature exists on all equipment. Otherwise, repeated S10,F3 messages should be used.

Structure: <TEXT>

S10, F10 Broadcast Acknowledge (BCA) *S, H <- E*

Description: Acknowledge or error.

Structure: <ACKC10> (1 byte binary: 0 = accepted for display,
1 = message will not be displayed)

Chapter 6
AdeptGEM Statements

This chapter describes the additional statements that are provided with the AdeptGEM module.

In addition to new statements there are two existing AIM statements that have been modified, IF and WHILE.

> **CAUTION:** AIM 3.0 allows you to create multiple copies of all databases so you can have a copy of data that is specific to a particular module. There is also a global copy of all databases if you want data to be shared among multiple modules. In the case of AdeptGEM databases, the data is global to the entire AIM system and identical data must be shared by all executing tasks. Thus, you can not add any AdeptGEM databases to any module. All AdeptGEM databases must be accessed globally.

6.1 GEM Statements

The following statements are added to the AIM system by the AdeptGEM module.

DISABLE_COMM

This routine will disable host communication and place the equipment in the "not communicating" state. See **page 125** for details.

GENERATE_ALARM

This statement sends an alarm message to the host. The syntax is:

```
GENERATE_ALARM --gemitem--
```

A `--gemitem--` of type Alarm, indicates the alarm that will be sent. See **page 126** for details.

GENERATE_EVENT

This statement sends a collection event to the host. The syntax is:

```
GENERATE_EVENT --gemitem--
```

A `--gemitem--` of type Collection Event, indicates which CE will be sent. See **page 127** for details.

LINE_CHECK

This statement will send an S1F13 message. If the equipment and host are online and communicating, --gemvariable-- will receive a value of 0 (success); otherwise, it will receive a standard AIM error code. See **page 182** for details.

```
      LINE_CHECK --gemvariable-- = COMM_STATUS
```

RUN_CHECK

This routine is used in a sequence to verify that the current state of the equipment is compatible with host control of robot motion. The syntax is:

```
      RUN_CHECK --gemvariable-- = MOTION_OK
```

NOTE: This statement can be used only in AIM control sequences.

See **page 184** for details.

SEND_HOST_MSG

This statement sends a text message to the host. The syntax is:

```
      SEND_HOST_MSG --gemvariable--
```

A `--gemvariable--` of type string variable or string function, indicates the string that will be sent. See **page 205** for details.

SET_STATE

This statement sets a variable to a state. The syntax is:

```
      SET_STATE --gemitem--
```

The --gemitem-- record must be of type State variable.

Note also that the SET and SETS statements can change Status Variables in the GEM Variables database.

The various control statements (IF, FOR, etc.) can access the GEM Variables database. See **page 206** for details.

Chapter 7
The AdeptGEM Databases

This chapter describes the use and format of the databases included in the AdeptGEM module. The databases specific to this module are:

- The GEM Variables Database
- The GEM Items Database

Unlike standard AIM databases, AdeptGEM databases may be accessed by the host as well as an executing sequence. In order to control simultaneous access to the databases, special locks have been installed. If you create custom code to access the AdeptGEM databases, make sure you use the routines described in **Chapter 9** and not the “db.” routines described in the AIM reference guides.

> **NOTE:** When communication is enabled, the GEM databases can still be accessed from the user interface, but the operator may be prevented from making changes to the data.

> **CAUTION:** Some of the database records are “predefined” records. There are two types of predefined records: records that should not be deleted, but with values you may need to change; and records that should not be deleted or changed. The predefined records are described with each database. Be careful to not make inappropriate changes to these records.

> **CAUTION:** AIM 3.x allows you to create multiple copies of all databases so you can have a copy of data that is specific to a particular module. There is also a global copy of all databases if you want data to be shared among multiple modules. In the case of AdeptGEM databases, the data is global to the entire AIM system and identical data must be shared by all executing tasks. Thus, you can not add any AdeptGEM databases to any module. All AdeptGEM databases must be accessed globally.

In addition, the following database management restrictions apply:

- Edit operations (New, Cut, Copy, Paste) are not permitted if GEM communications are enabled.
- Predefined records (i.e., ID’s 9000 - 9999) cannot be cut.
- Changing an Item ID or record type is not permitted if GEM communications are enabled.
- The value in the Item ID field cannot be deleted.
- When the ID of an item or variable, or the record type of an item is changed, the database is sorted (in memory only) to position the record in the correct order.

7.1 The GEM Variables Database

The GEM Variables database stores equipment constants (ECs), status variables (SVs), and data values (DVVALs) used in the GEM environment. There are several classes of variables: predefined variables that the user sets, predefined variables that can be changed only by the equipment, and user-defined variables:

- Status Variable (SV)

  Status variables are used to communicate the status of the workcell (i.e., number of widgets completed, number of component parts remaining, etc.). These are values that are set by the equipment but are read-only to the host (see the table below). To comply with the SEMI standards, you must guarantee that these values are always kept current.

  For example, you could have a sensor at the end of your circuit board assembly line that is tripped each time a completed assembly passes by. This action causes the total count to be increased one unit. This value (the total count) is stored in a status variable that is updated each time the sensor is tripped. It must be kept current so that anyone reading this value will know the exact number of assemblies that have been completed.

  Status variables include user defined variables that may be input signals, output signals, AIM variables, **ai.ctl[ ]** values, or state variables.

  There are also a number of predefined GEM internal status variables that cannot be modified.

- Equipment Constant (EC)

  Equipment constants provide a way to store values for equipment parameters. These values control aspects of equipment behavior (i.e., how many widgets to build, what parts to use for a widget assembly, etc.). Remember that these values *cannot* be changed by the equipment when it is in the on-line state (see the table below).

  There are many cases where you will be defining your own equipment constants. For example, you may want to customize the equipment start-up configuration for a particular process or part.

  There are also many predefined equipment constant records that can be modified but not deleted. Some of these can be changed by both the host and the equipment operator and some can be changed only by the equipment operator.

- Data Value (DVVAL)

  Data values are values that are guaranteed to be accurate only after a specific Collection Event has been generated. This method eliminates the need for constant polling of the equipment, and reduces the overhead on the system. For example, these values could be used to store yield data from a Statistical Process Control (SPC) application.

  These values can be modified by the equipment but are read-only the host (see the table below). DVVALs can be read by the host at any time. However, the DVVALs are only guaranteed to be valid after a Collection Event occurs.

  Data values can include user defined variables (defined as a String function or Real function) in the GEM Variables database. The user also needs to modify the routine **gm.user.packval( )**.

  There are also predefined internal GEM variables that cannot be modified.

The table below provides a summary of the Read/Write privileges for the Equipment and Host:

Variable Type	On-line & Process Related	Equipment	Host
Status Variables and	Yes	Read/Write	Read only
	No	Read/Write	Read only
Equipment Constants	Yes	Read only	Read/Write
	No	Read/Write	Read/Write
Data Value	N/A	Read/Write	Read only

An Example of SVs, ECs, and DVVALs

The use of status variables (SVs), equipment constants (ECs), and data values (DVVALs) is illustrated in the following scenario:

- It is 8:00 A.M. The production schedule calls for one 1200-piece batch of Widget X to be made. The workcell will start this build on the first shift and continue building this assembly until the batch is completed. The batch size value (1200 pieces) for Widget X is sent to the workcell and stored as an equipment constant.
- It is now 6:30 P.M. and the second shift is continuing the build of Widget X. The production supervisor wants to know how may assemblies have been completed. He goes to the host and checks the Total Widgets Completed field. The workcell has completed 652 widgets. He stays at the host computer for three more minutes and sees the count updated three times. (Remember, this value is stored in a status variable so it must be guaranteed to *always* show the current total.)
- While the second shift supervisor is at the host computer, he also decides to check the Average Pieces / Hour field. This field displays an average rate of 61 pieces per hour. It also indicates that this value was last updated at 6:00 P.M. Since it is now 6:33 P.M., he knows that this may not be the current average. (Remember, this type of information is typically stored as a data value. It is only guaranteed to be current *after* a specific Collection Event has been generated.)
- It is now 5:45 A.M. and the third shift is on duty. As the 1200th Widget X rolls off the production line, the Total Widgets Completed field is immediately updated (Status Value). Since the batch is now completed, the statistics for the run are calculated and stored as data values, and a Collection Event is generated. The data values will be included in a production report for the morning production meeting.

Variable IDs (VIDs)

Variable IDs (VIDs) are a class of SEMI variables composed of equipment constants (ECs), status variables (SVs), and data values (DVVALs). VIDs are used extensively by AdeptGEM. These IDs are integer values in the range 1 to 32,767 with the range from 9000 to 9999 reserved for AdeptGEM defined VIDs. VIDs are assigned to records in the GEM Variables database. Each record has a unique Variable ID number.

SEMI Data Types

The host data types come from a list of data types defined in the SEMI E5-95 standard. The following table provide a list of the available host data types and indicates those that are not supported:

Table 7-1
Host Data Types

Octal	Meaning	Supported (Y/N)
00	LIST (length in elements)	Y
10	Binary	Y
11	Boolean	Y
20	ASCII[1]	Y
21	JIS-8	N
30	8-byte integer (signed)[2]	N
31	1-byte integer (signed)	Y
32	2-byte integer (signed)[2]	Y
34	4-byte integer (signed)[2]	Y
40	8-byte floating point[3]	N
44	4-byte floating point[3]	Y
50	8-byte integer (unsigned)[2]	N
51	1-byte integer (unsigned)	Y
52	2-byte integer (unsigned)[2]	Y
54	4-byte integer (unsigned)[2]	Y

Notes:

(1) Non-printing characters are equipment specific.

(2) Most significant byte sent first.

(3) IEEE 754. The byte containing the sign bit is sent first.

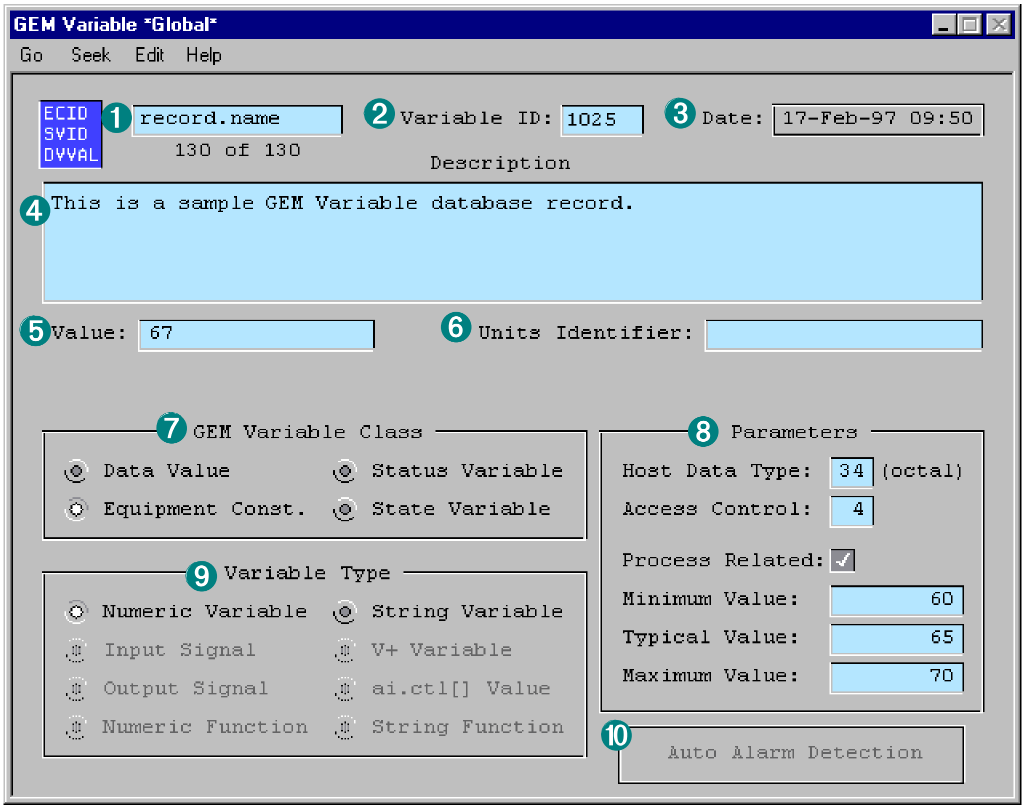
The GEM Variables Menu Page

The GEM Variables menu page is used to create records of equipment constants (ECs), status variables (SVs), and data values (DVVALs). The records are stored in the GEM Variables database.

To create a GEM Variables database record:

Edit ➡ GEM Items ➡ Edit ➡ New Record

A new record is displayed. Figure 7-1 shows an example of a completed record.



record.name

130 of 130

Variable ID: 1025

Date: 17-Feb-97 09:50

Description

This is a sample GEM Variable database record.

Value: 67

Units Identifier:

GEM Variable Class

○ Data Value ○ Status Variable

● Equipment Const. ○ State Variable

Variable Type

● Numeric Variable ○ String Variable

○ Input Signal ○ V+ Variable

○ Output Signal ○ ai.ctl[] Value

○ Numeric Function ○ String Function

Parameters

Host Data Type: 34 (octal)

Access Control: 4

Process Related: ☑

Minimum Value: 60

Typical Value: 65

Maximum Value: 70

Figure 7-1
GEM Variables Menu Page

❶ The record name.

❷ The Variable ID number.

❸ Displays the date and time that the record was created or modified.

❹ Optional description of the record.

❺ Contains the value of the record. (This item may be user defined depending on the selections in items ❼ and ❾).

❻ A units identifier that will be sent to the host when the Variable ID is reported. (This item does not appear for some selection in items ❼ and ❾).

❼ This group is used to set the GEM Variables class for the record. Each class is explained in the following sections.

❽ This group is used to define the parameters for the variable. (The items in this group may vary depending on the selections in items ❼ and ❾.)

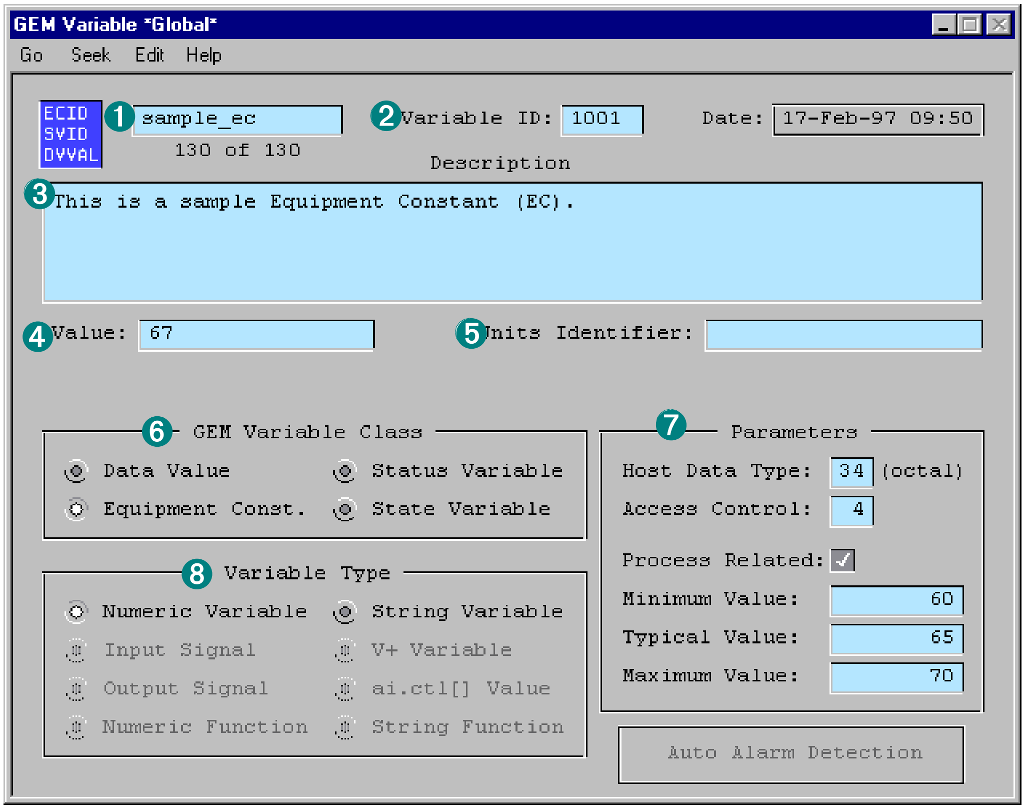
- Host Data Type: specifies the data type used by this variable
- Access Control: specifies the access level required to modify this record
- Process Related: when checked, indicates this is a process related variable. See section 3.10 on page 36 for details.
- Minimum Value, Typical Value, Maximum Value: specifies the expected range of values for this variable.

❾ This group is used to set the variable type for the record.

❿ This button activates automatic alarm detection. See section 3.8 on page 32 for details.

Equipment Constant

The following figure shows an equipment constant in the GEM Variables database:



sample_ec

130 of 130

Variable ID: 1001

Date: 17-Feb-97 09:50

Description

This is a sample Equipment Constant (EC).

Value: 67

Units Identifier:

GEM Variable Class

○ Data Value ○ Status Variable

● Equipment Const. ○ State Variable

Variable Type

● Numeric Variable ○ String Variable

○ Input Signal ○ V+ Variable

○ Output Signal ○ ai.ctl[] Value

○ Numeric Function ○ String Function

Parameters

Host Data Type: 34 (octal)

Access Control: 4

Process Related: ☑

Minimum Value: 60

Typical Value: 65

Maximum Value: 70

Figure 7-2
GEM Variables Database, Equipment Constant

To add a new equipment constant:

1. Create a new GEM Variables record.
2. Enter a name for the new record (item ❶).
3. Assign a Variable ID (item ❷). See “Variable IDs (VIDs)” on page 87 for more details.
4. Enter a description for this record (item ❸).

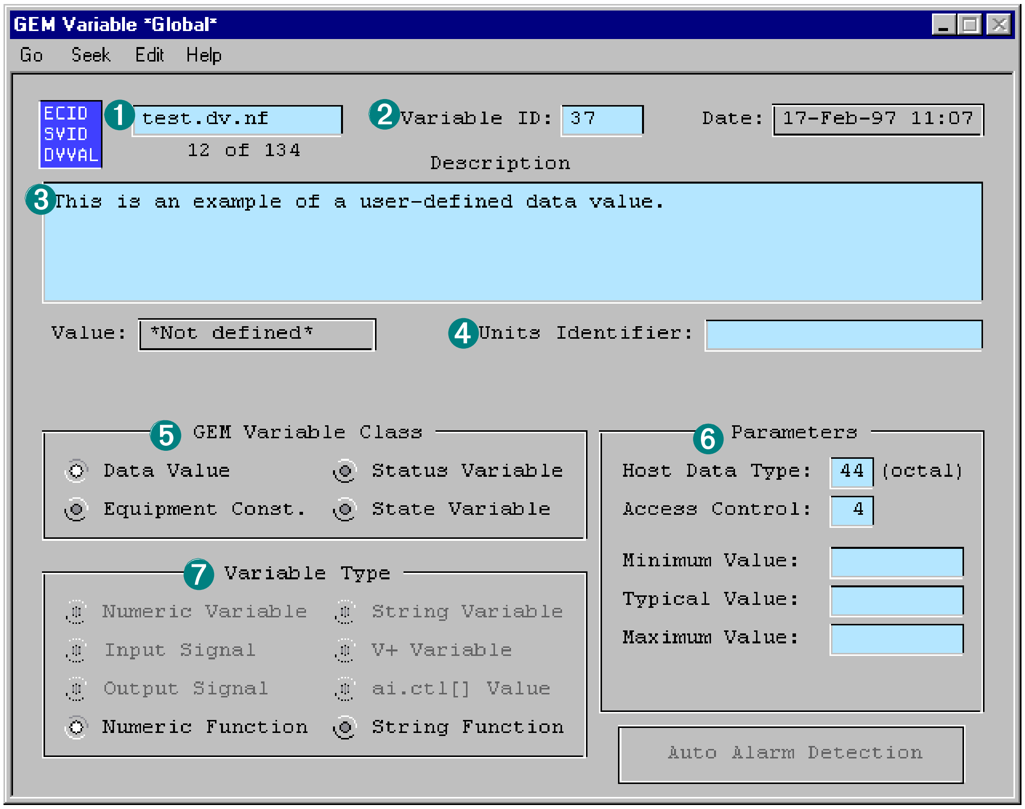
5. Select ◉ **Equipment Const.** as the GEM Variable Class (item ❻).
6. Select ◉ **Numeric Variable** or ◉ **String Variable** as the Variable Type (item ❽).
7. Specify a Value for the variable (item ❹).
8. If a numeric variable is being defined, enter a units identifier (item ❺)that will be sent to the host when this Variable ID is reported. The units identifier should correspond to the identifiers described in Section 9 "Units of Measure" in SEMI E5-95.

Refer to item ❼ for the following steps:

9. Specify a Host Data Type. See **Table 7-1** for the list of data types.
10. Specify the Access Control. Enter a single digit (0 – 4) in this field to specify the user access level required to change the record. Insert a 1 (no modification allowed) or 0 (modification allowed) in front of the access digit to control host modification of the EC.
11. Select Process Related to prevent this ECID from being updated when a process is running.
12. Enter the Minimum, Maximum, and Typical values as appropriate.

Data Value

The next figure shows a data value (DVVAL) in the GEM Variables database:



test.dv.nf

12 of 134

Variable ID: 37

Date: 17-Feb-97 11:07

Description

This is an example of a user-defined data value.

Value: *Not defined*

Units Identifier:

GEM Variable Class

○ Data Value ○ Status Variable
○ Equipment Const. ○ State Variable

Parameters

Host Data Type: 44 (octal)
Access Control: 4
Minimum Value:
Typical Value:
Maximum Value:

Variable Type

○ Numeric Variable ○ String Variable
○ Input Signal ○ V+ Variable
○ Output Signal ○ ai.ctl[] Value
○ Numeric Function ○ String Function

Auto Alarm Detection

Figure 7-3
GEM Variables Database, Data Value

There are a number of predefined data values (DVVALs) in the AdeptGEM Variables database that cannot be changed. The functions of the predefined DVVALs are described in **Table 7-2**.

The user can also define new DVVALs. These are defined as a Numeric Function or String Function using the menu page shown in Figure 7-3.

To define a new DVVAL:

1. Create a new GEM Variables record.
2. Enter a name for the new record (item ❶).
3. Assign a Variable ID (item ❷). See "Variable IDs (VIDs)" on page 87 for more details.
4. Enter a description of this record (item ❸).
5. Select ● **Data Value** as the GEM Variable Class (item ❺).
6. Select ● **Numeric Function** or ● **String Function** as the Variable Type (item ❼).
7. If a numeric variable is being defined, enter a units identifier (item ❹)that will be sent to the host when this Variable ID is reported. The units identifier should correspond to the identifiers described in Section 9 "Units of Measure" in SEMI E5-95.

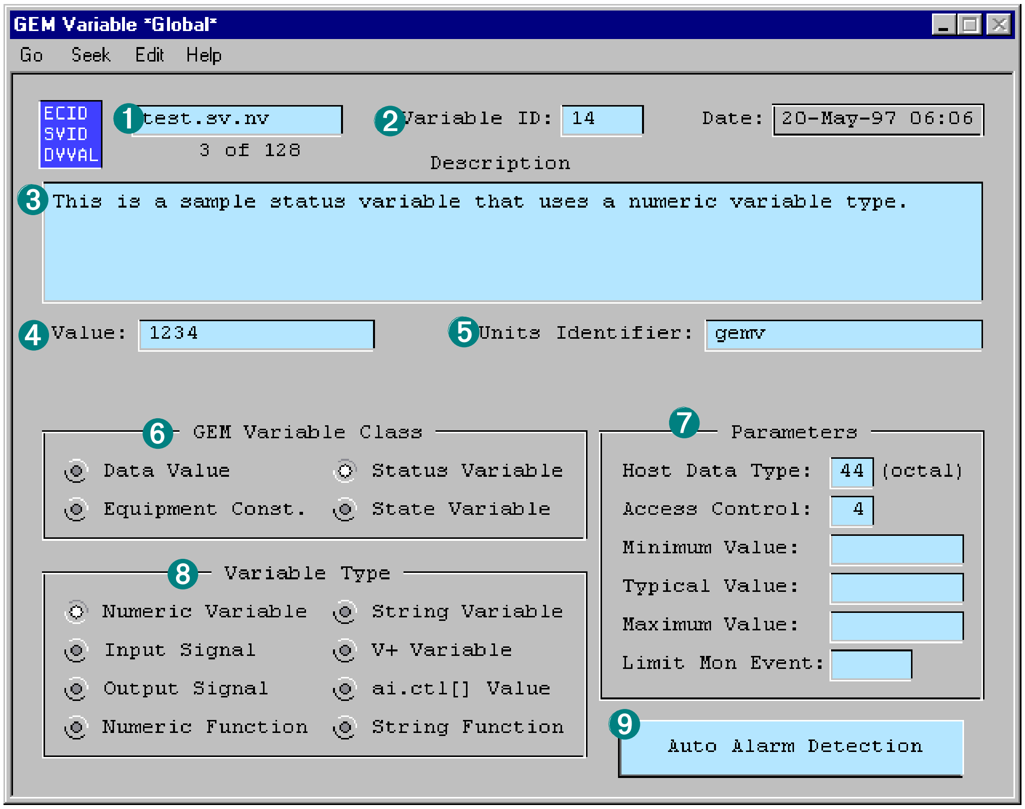
Refer to item ❻ for the following steps:

8. Specify a Host Data Type. See Table 7-1 for the list of data types.
9. Specify the Access Control. Enter a single digit (0 – 4) in this field to specify the user access level required to change the record. Insert a 1 (no modification allowed) or 0 (modification allowed) in front of the access digit to control host modification of the EC.
10. Enter the Minimum, Maximum, and Typical values as appropriate.

In addition to creating the database record for the DVVAL, you must also modify the V$^+$ routine **gm.user.packval( )** as required to return the value of the DVVAL. See the description of **gm.user.packval( )** on page 171 for more information.

Status Variable Using a Numeric Variable

The next figure shows a status variable in the GEM Variables database that uses a numeric value in its definition:



test.sv.nv

3 of 128

Variable ID: 14

Date: 20-May-97 06:06

Description

This is a sample status variable that uses a numeric variable type.

Value: 1234

Units Identifier: gemv

GEM Variable Class

○ Data Value ● Status Variable

○ Equipment Const. ○ State Variable

Variable Type

● Numeric Variable ○ String Variable

○ Input Signal ○ V+ Variable

○ Output Signal ○ ai.ctl[] Value

○ Numeric Function ○ String Function

Parameters

Host Data Type: 44 (octal)

Access Control: 4

Minimum Value:

Typical Value:

Maximum Value:

Limit Mon Event:

Figure 7-4
GEM Variables Database, Status Variable Using a Numeric Variable

To define a new status variable that uses a numeric variable:

1. Create a new GEM Variables record.
2. Enter a name for the new record (item ❶).
3. Assign a Variable ID (item ❷). See **“Variable IDs (VIDs)” on page 87**.
4. Enter an optional description of this record (item ❸).
5. Select ◉ **Status Variable** as the GEM Variables class (item ❻).
6. Select ◉ **Numeric Variable** as the Variable Type (item ❽).
7. Specify a Value for the variable (item ❹).
8. Enter a units identifier (item ❺)that will be sent to the host when this Variable ID is reported. The units identifier should correspond to the identifiers described in Section 9 “Units of Measure” in SEMI E5-95.

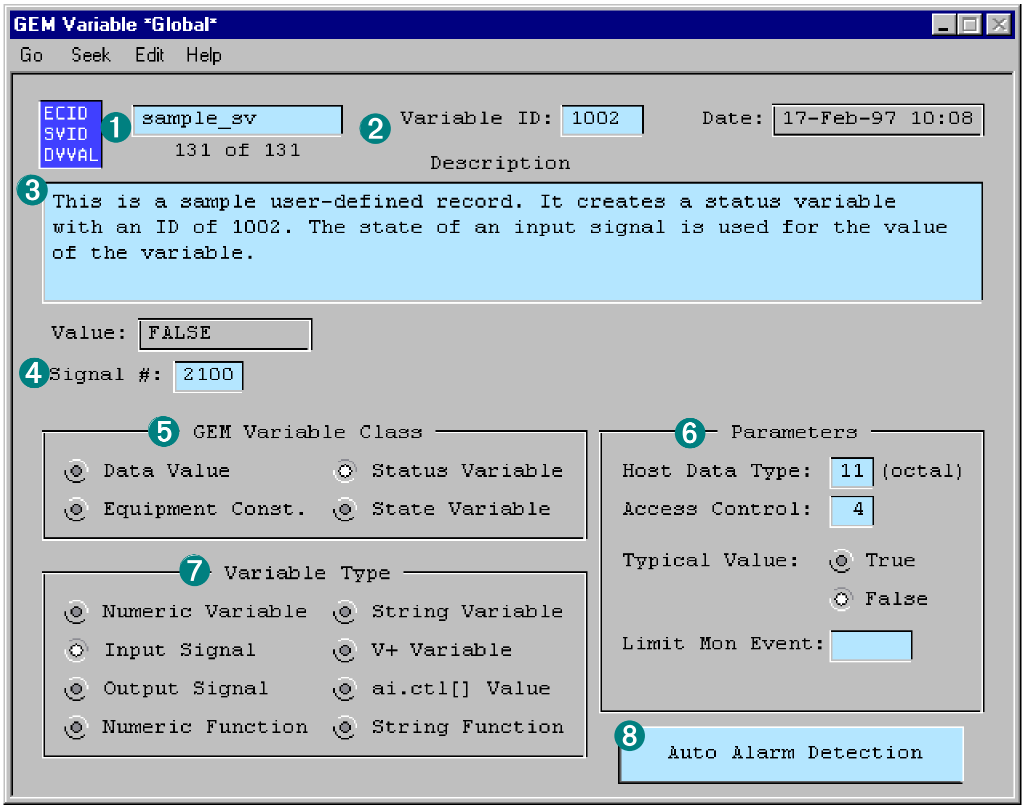
Refer to item ❼ for steps 9 through 12:

9. Specify a Host Data Type. See **Table 7-1, “Host Data Types”, on page 88**.

10. Specify the Access Control. Enter a single digit (0 – 4) in this field to specify the user access level required to change the record. Insert a 1 (no modification allowed) or 0 (modification allowed) in front of the access digit to control host modification of the record.
11. Enter the Minimum, Maximum, and Typical values as appropriate.
12. If this record relates to a limit, enter the number of the limits monitoring event. This number is a Collection Event ID in the Collect Event record type. See section 3.6.
13. If this record will define an automatically detected alarm, press **Auto Alarm Detection** (item ❾). The options described in **"Automatic Alarm Detection" on page 33** will be displayed.

Status Variable Using a Digital I/O Signal

The next figure shows a status variable (SV) in the GEM Variables database that uses a digital I/O signal:



sample_sv

131 of 131

Variable ID: 1002

Date: 17-Feb-97 10:08

Description

This is a sample user-defined record. It creates a status variable with an ID of 1002. The state of an input signal is used for the value of the variable.

Value: FALSE

Signal #: 2100

GEM Variable Class

◎ Data Value ○ Status Variable
◎ Equipment Const. ◎ State Variable

Variable Type

◎ Numeric Variable ◎ String Variable
○ Input Signal ◎ V+ Variable
◎ Output Signal ◎ ai.ctl[] Value
◎ Numeric Function ◎ String Function

Parameters

Host Data Type: 11 (octal)

Access Control: 4

Typical Value: ◎ True ○ False

Limit Mon Event:

Figure 7-5
GEM Variables Database, Status Variable ID Using Digital I/O

To define a new status variable that uses a digital I/O signal as its value:

1. Create a new GEM Variables record.
2. Enter a name for the record (item ❶).
3. Assign a Variable ID (item ❷). See **"Variable IDs (VIDs)" on page 87**.
4. Enter a description of this record (item ❸).

5. Select ◉ **Status Variable** as the GEM Variable Class (item ❺).
6. Select ◉ **Input Signal** or ◉ **Output Signal** as the Variable Type (item ❼).
7. Specify the Signal number (item ❹).

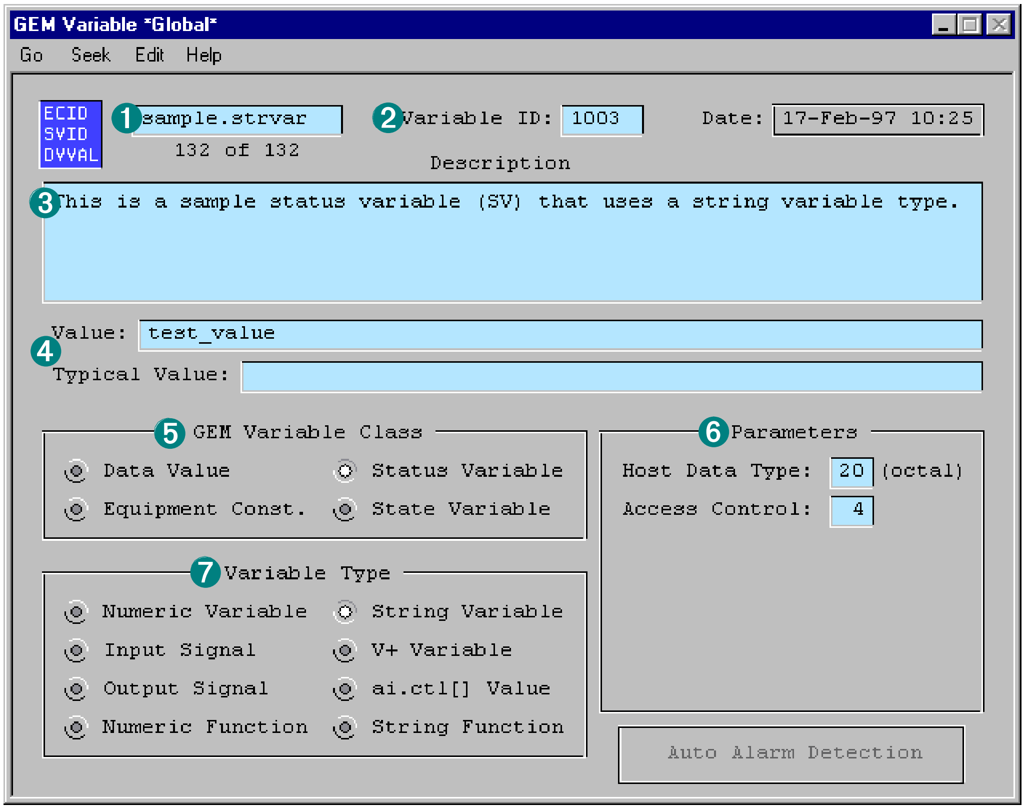
Refer to item ❻ for steps 8 through 11:

8. Specify a Host Data Type. See **Table 7-1, “Host Data Types”, on page 88**—the most appropriate data type is Boolean, type 11.
9. Specify the Access Control. Enter a single digit (0 – 4) in this field to specify the user access level required to change the record. Insert a 1 (no modification allowed) or 0 (modification allowed) in front of the access digit to control host modification of the record.
10. Select ◉ **True** or ◉ **False** as the Typical Value for the signal.
11. If this record relates to a limit, enter the number of the limits monitoring event. This number is a Collection Event ID in the Collect Event record type. See section **3.6**.
12. If this record will define an automatically detected alarm, press **Auto Alarm Detection** (item ❽). The options described in **“Automatic Alarm Detection” on page 33** will be displayed.

Note that this record specifies the digital signal to get a value from, not the actual state of the signal. The state of the signal is returned as the value of the record.

Status Variable Using a String Variable

The next figure shows a status variable in the GEM Variables database that uses a string variable type in its definition:



sample.strvar

132 of 132

Variable ID: 1003

Date: 17-Feb-97 10:25

Description

This is a sample status variable (SV) that uses a string variable type.

Value: test_value

Typical Value:

GEM Variable Class

○ Data Value ○ Status Variable

○ Equipment Const. ○ State Variable

Parameters

Host Data Type: 20 (octal)

Access Control: 4

Variable Type

○ Numeric Variable ○ String Variable

○ Input Signal ○ V+ Variable

○ Output Signal ○ ai.ctl[] Value

○ Numeric Function ○ String Function

Figure 7-6
GEM Variables Database, Status Variable Using a String Variable

To define a new status variable that uses a string variable:

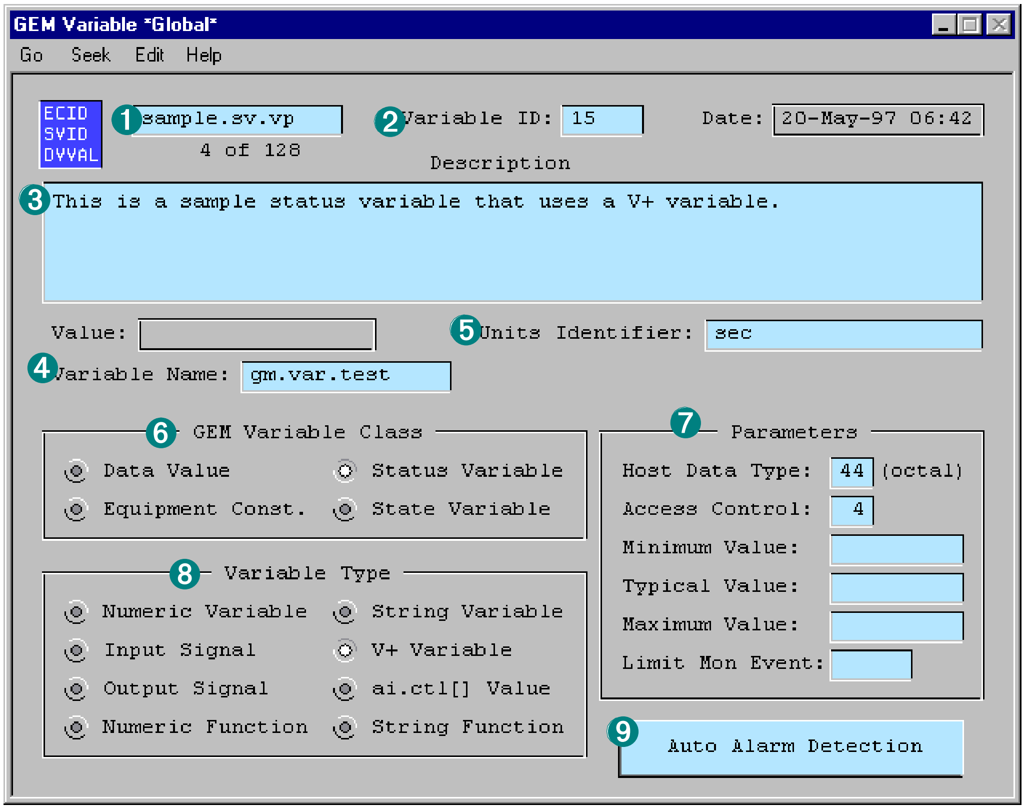
1. Create a new GEM Variables record.
2. Enter a name for the new record (item ❶).
3. Assign a Variable ID (item ❷). See **“Variable IDs (VIDs)” on page 87**.
4. Enter a description of this record (item ❸).
5. Select ● **Status Variable** as the GEM Variable Class (item ❺).
6. Select ● **String Variable** as the Variable Type (item ❼).
7. Specify a Value for the variable (item ❹). You can also specify an optional Typical Value.

Refer to item ❻ for the following steps:

8. Specify a Host Data Type. See **Table 7-1, “Host Data Types”, on page 88**.
9. Specify the Access Control. Enter a single digit (0 – 4) in this field to specify the user access level required to change the record. Insert a 1 (no modification allowed) or 0 (modification allowed) in front of the access digit to control host modification of the record.

Status Variable Using a V+ Variable

The next figure shows a status variable in the GEM Variables database that uses a V+ variable in its definition:



sample.sv.vp
4 of 128
Variable ID: 15
Date: 20-May-97 06:42
Description
This is a sample status variable that uses a V+ variable.
Value:
Units Identifier: sec
Variable Name: gm.var.test

GEM Variable Class
○ Data Value ● Status Variable
○ Equipment Const. ○ State Variable

Variable Type
○ Numeric Variable ○ String Variable
○ Input Signal ● V+ Variable
○ Output Signal ○ ai.ctl[] Value
○ Numeric Function ○ String Function

Parameters
Host Data Type: 44 (octal)
Access Control: 4
Minimum Value:
Typical Value:
Maximum Value:
Limit Mon Event:

Figure 7-7
GEM Variables Database, Status Variable Using a V+ Variable

To define a new status variable that uses a V+ variable:

1. Create a new GEM Variables record.
2. Enter a name for the new record (item ❶).
3. Assign a Variable ID (item ❷). See “Variable IDs (VIDs)” on page 87.
4. Enter an optional description of this record (item ❸).
5. Select ● **Status Variable** as the GEM Variables class (item ❻).
6. Select ● **V+ Variable** as the Variable Type (item ❽).
7. Specify a name for the variable (item ❹).
8. Enter a units identifier (item ❺)that will be sent to the host when this Variable ID is reported. The units identifier should correspond to the identifiers described in Section 9 “Units of Measure” in SEMI E5-95.

Refer to item ❼ for steps 9 through 12:

9. Specify a Host Data Type. See Table 7-1, “Host Data Types”, on page 88.

10. Specify the Access Control. Enter a single digit (0 – 4) in this field to specify the user access level required to change the record. Insert a 1 (no modification allowed) or 0 (modification allowed) in front of the access digit to control host modification of the record.

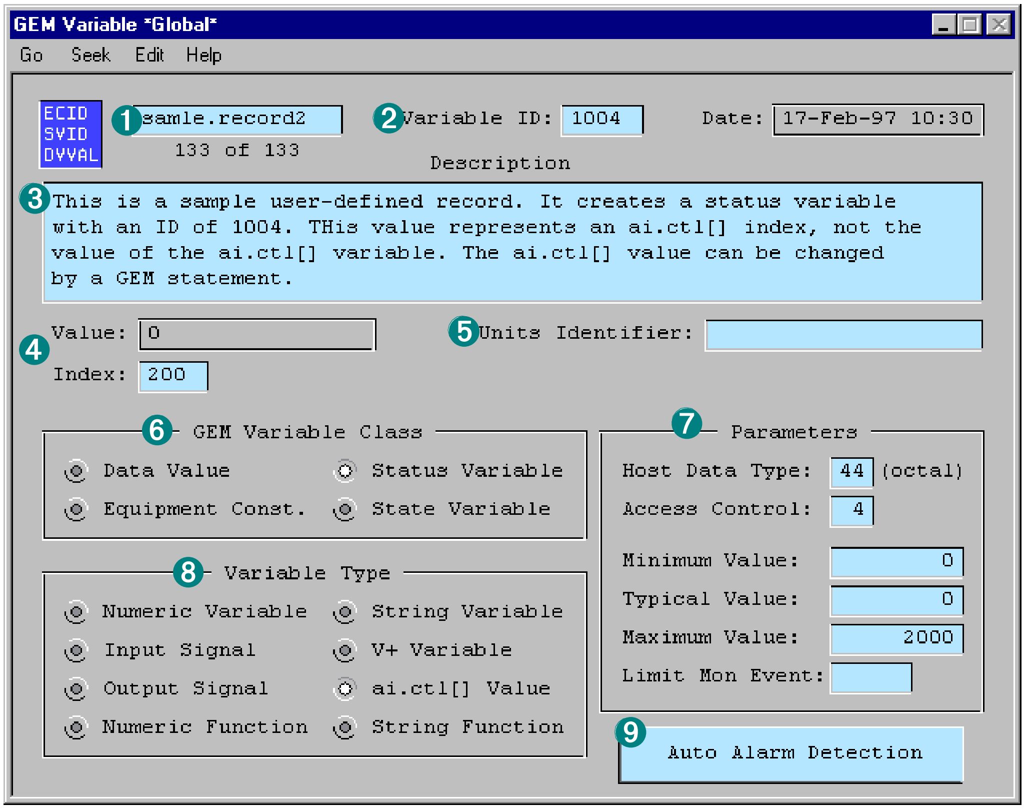
11. Enter the Minimum, Maximum, and Typical values as appropriate.

12. If this record relates to a limit, enter the number of the limits monitoring event. This number is a Collection Event ID in the Collect Event record type. See section 3.6.

13. If this record will define an automatically detected alarm, press **Auto Alarm Detection** (item ❾). The options described in **"Automatic Alarm Detection" on page 33** will be displayed.

Status Variable Using an ai.ctl[ ] Value

The next figure shows a status variable in the GEM Variables database that uses an **ai.ctl[ ]** value in its definition:



samle.record2

133 of 133

Variable ID: 1004

Date: 17-Feb-97 10:30

Description

This is a sample user-defined record. It creates a status variable with an ID of 1004. THis value represents an ai.ctl[] index, not the value of the ai.ctl[] variable. The ai.ctl[] value can be changed by a GEM statement.

Value: 0

Units Identifier:

Index: 200

GEM Variable Class

○ Data Value ● Status Variable
○ Equipment Const. ○ State Variable

Variable Type

○ Numeric Variable ○ String Variable
○ Input Signal ○ V+ Variable
○ Output Signal ● ai.ctl[] Value
○ Numeric Function ○ String Function

Parameters

Host Data Type: 44 (octal)
Access Control: 4
Minimum Value: 0
Typical Value: 0
Maximum Value: 2000
Limit Mon Event:

Figure 7-8
GEM Variables Database, Status Variable Using an ai.clt[ ] Value

To define a new status variable that uses an ai.clt[ ] value:

1. Create a new GEM Variables record.

2. Enter a name for the new record (item ❶).

3. Assign a Variable ID (item ❷). See **"Variable IDs (VIDs)" on page 87**.

4. Enter an optional description of this record (item ❸).

5. Select ◉ **Status Variable** as the GEM Variables class (item ❻).
6. Select ◉ **ai.ctl[ ] Value** as the Variable Type (item ❽).
7. Specify an Index for the variable (item ❹).
8. Enter a units identifier (item ❺)that will be sent to the host when this Variable ID is reported. The units identifier should correspond to the identifiers described in Section 9 “Units of Measure” in SEMI E5-95.

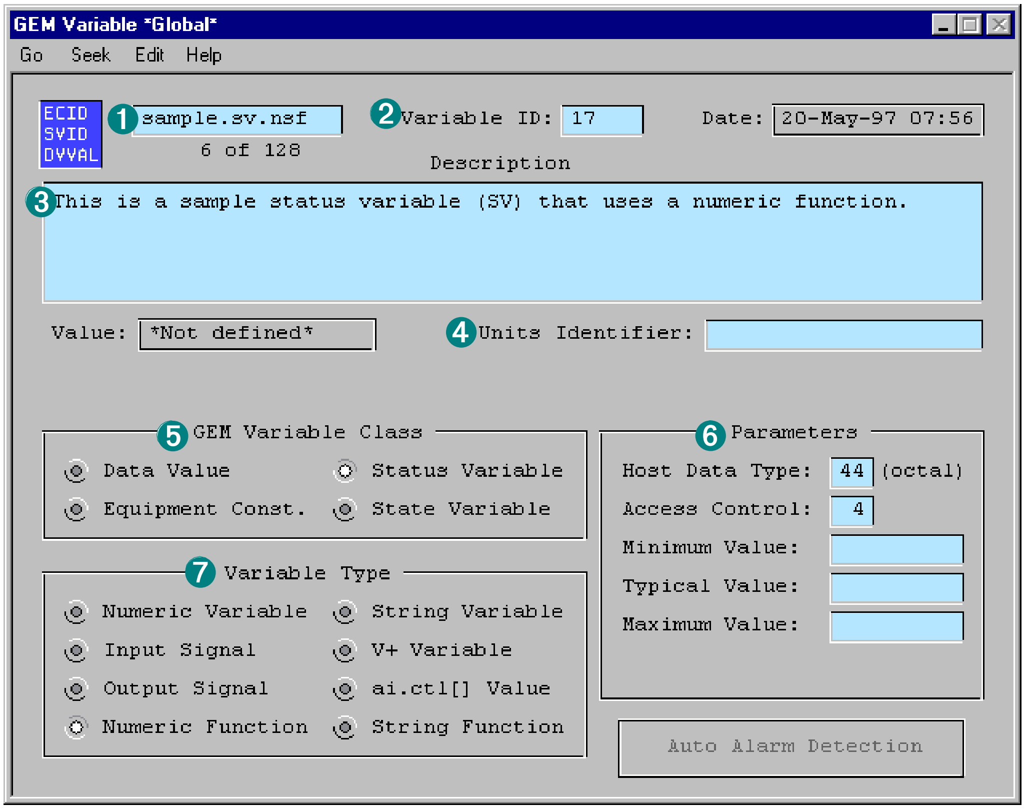
Refer to item ❼ for steps 9 through 12:

9. Specify a Host Data Type. See **Table 7-1, “Host Data Types”, on page 88**.
10. Specify the Access Control. Enter a single digit (0 – 4) in this field to specify the user access level required to change the record. Insert a 1 (no modification allowed) or 0 (modification allowed) in front of the access digit to control host modification of the record.
11. Enter the Minimum, Maximum, and Typical values as appropriate.
12. If this record relates to a limit, enter the number of the limits monitoring event. This number is a Collection Event ID in the Collect Event record type. See section **3.6**.
13. If this record will define an automatically detected alarm, press **Auto Alarm Detection** (item ❾). The options described in **“Automatic Alarm Detection” on page 33** will be displayed.

Note that this record specifies the **ai.ctl[ ]** to get a value from, not the actual value of the **ai.ctl[ ]** array element. The value of the **ai.ctl[ ]** element is returned as the value of the record.

Status Variable Using a Numeric or String Function

The next figure shows a status variable in the GEM Variables database that uses a numeric function:



sample.sv.nsf

6 of 128

Variable ID: 17

Date: 20-May-97 07:56

Description

This is a sample status variable (SV) that uses a numeric function.

Value: *Not defined*

Units Identifier:

GEM Variable Class

○ Data Value
● Status Variable
○ Equipment Const.
○ State Variable

Variable Type

○ Numeric Variable
○ String Variable
○ Input Signal
○ V+ Variable
○ Output Signal
○ ai.ctl[] Value
● Numeric Function
○ String Function

Parameters

Host Data Type: 44 (octal)

Access Control: 4

Minimum Value:

Typical Value:

Maximum Value:

Figure 7-9
GEM Variables Database, Data Value

To define a new status variable that uses a numeric or string function:

1. Create a new GEM Variables record.
2. Enter a name for the new record (item ❶).
3. Assign a Variable ID (item ❷). See **“Variable IDs (VIDs)” on page 87** for more details.
4. Enter a description of this record (item ❸).
5. Select ◉ **Status Variable** as the GEM Variable Class (item ❺).
6. Select ◉ **Numeric Function** or ◉ **String Function** as the Variable Type (item ❼).
7. If a numeric variable is being defined, enter a units identifier (item ❹)that will be sent to the host when this Variable ID is reported. The units identifier should correspond to the identifiers described in Section 9 “Units of Measure” in SEMI E5-95.

Refer to item ❻ for the following steps:

8. Specify a Host Data Type. See **Table 7-1** for the list of data types.

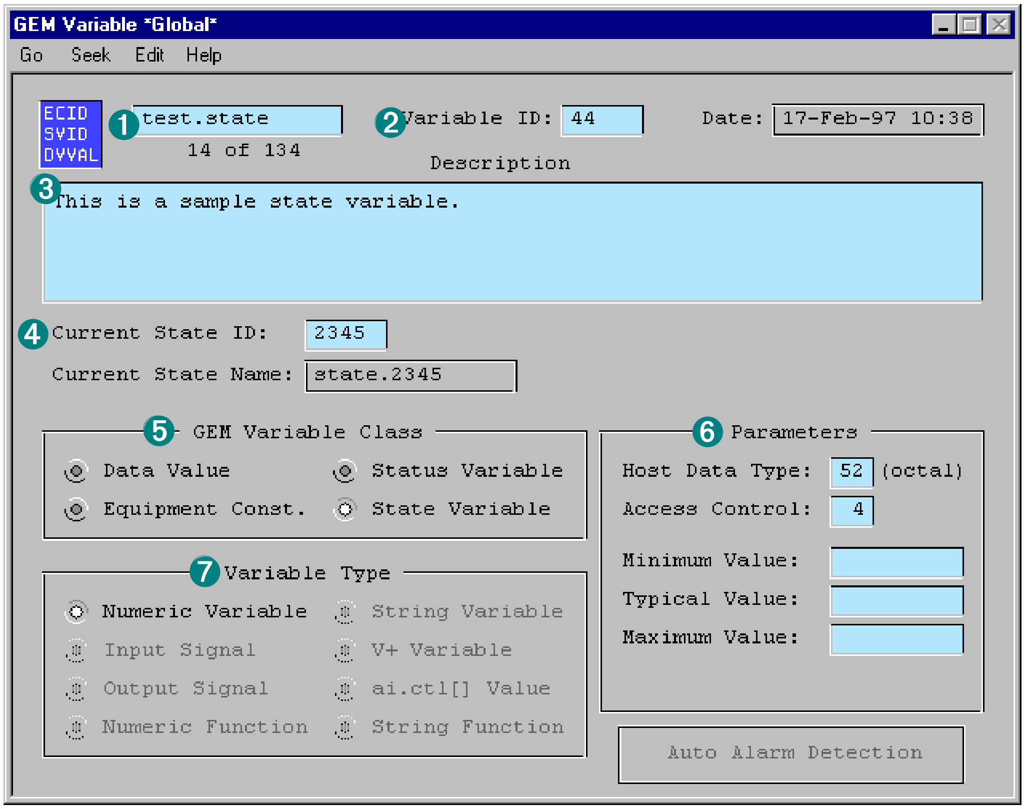
9. Specify the Access Control. Enter a single digit (0 – 4) in this field to specify the user access level required to change the record. Insert a 1 (no modification allowed) or 0 (modification allowed) in front of the access digit to control host modification of the EC.

10. Enter the Minimum, Maximum, and Typical values as appropriate.

In addition to creating the database record for the status variable, you must also modify the V+ routine **gm.user.packval( )** as required to return the value of the status variable. See the description of **gm.user.packval( )** on page 171 for more information.

State Variable

The next figure shows an example definition for a State Variable:



test.state
14 of 134

Variable ID: 44

Date: 17-Feb-97 10:38

Description
This is a sample state variable.

Current State ID: 2345

Current State Name: state.2345

GEM Variable Class
○ Data Value ○ Status Variable
○ Equipment Const. ● State Variable

Variable Type
● Numeric Variable ○ String Variable
○ Input Signal ○ V+ Variable
○ Output Signal ○ ai.ctl[] Value
○ Numeric Function ○ String Function

Parameters
Host Data Type: 52 (octal)
Access Control: 4
Minimum Value:
Typical Value:
Maximum Value:

Figure 7-10
GEM Variables Database, State Variable

A State Variable is a special case of a Status Variable. This record works with State record types in the GEM Items database to hold the current state of user defined state models.

To define a new State Variable:

1. Create a new GEM Variables record.

2. Enter a name for the new record (item ❶).

3. Assign a Variable ID (item ❷). See **“Variable IDs (VIDs)” on page 87**.

4. Enter a description of this record (item ❸).

5. Select ● **State Variable** as the GEM Variable Class (item ❺).

6. The Variable Type (item ❼) ◉ **Numeric Variable** is automatically selected.
7. Enter the number of the Current State ID (item ❹). This number corresponds to the State ID field in a State record type. When the Current State ID number is entered, the name of the associated State record type is displayed in the Current State Name field.

Refer to item ❻ for the following steps:

8. Specify a Host Data Type. See **Table 7-1, "Host Data Types", on page 88**.
9. Specify the Access Control. Enter a single digit (0 – 4) in this field to specify the user access level required to change the record.
10. Enter the Minimum, Maximum, and Typical values as appropriate.

The value of this record represents a state ID from the state model record type. This ID represents the current state of a state model. The value 0 indicates that a state has not been defined for this state model. See **page 113** for details on the State record type.

Predefined Status Variables, Equipment Constants, and Data Values

The GEM Variables database contains a number of predefined records that support many of the features and requirements of the GEM specification. The following table describes these predefined records. The "Reference Name" is the name used in the SEMI standards. The "Description" gives a brief description of the variable, see the database records for additional details. Note, variables of type "ASCII" are string values in the GEM Variables database.

Table 7-2
Predefined SVs, ECs and DVVALs

ID	Reference Name	Class	Type	Description
9000	alarmsenabled	SV	List of unsigned 2-byte integers	List of ALIDs with the enabled option set.
9001	alarmsset	SV	List of unsigned 2-byte integers	List of ALIDs that are set (regardless of the setting of the enabled option).
9002	clock	SV	ASCII	yymmddhhmmsscc system time
9003	controlstate	SV	Unsigned 1-byte integer	Coded value representing the current Control state model: 0 = Communication disabled 1 = Equipment offline 2 = Attempt on-line 3 = Host offline 4 = On-line/local 5 = On-line/remote
9004	eventsenabled	SV	List of unsigned 2-byte integers	List of CEIDs with the event enabled option set.
9005	ppexecname	SV	ASCII	String that represents the currently selected AIM sequence (process program).

Table 7-2
Predefined SVs, ECs and DVVALs (Continued)

ID	Reference Name	Class	Type	Description
9006	prevprocstate (task 0)	SV	Unsigned 1-byte integer	Coded value representing the previous equipment process state (see section 2.3).
9007	processstate (task 0)	SV	Unsigned 1-byte integer	Coded value representing the current equipment process state (see section 2.3).
9008	commstat	SV	Unsigned 2-byte integers	An array of statistics for SECS-I/ HSMS-SS. The array will be empty if communication is not enabled.
9009	remote	SV	Boolean	True = Control state is remote False = Equipment is in local mode
9010	recent.debuglog	SV	ASCII	Holds the last 64 entries in the debug log. (ECID 9105 must have bit 4 set.)
9021 to 9074	ppstask1 pcstask1 ... ppstask27 pcstask27	SV	Unsigned 1-byte integer	Coded value representing the previous equipment process states for tasks 1 to 27 (see section 2.3)
9100	estcommtimeout	EC	Unsigned 2-byte integer	Interval between S1F13 attempts
9101	startcomen	EC	Boolean	True = Communications is enabled on start-up

9102	usersoftrev	EC	Boolean	<SOFTREV> type:
				TRUE = User defined (see ECID 9301)
				FALSE = V^+ version

9103	controlstartst	EC	Unsigned 1-byte integer	Control: Start-up state: 1 = Equip off-line 2 = Attempt on-line 3 = Host off-line 4 = On-line
9104	onlinefailstate	EC	Unsigned 1-byte integer	If attempt On-line fail state fails: 1 = Set equipment off-line state 3 = Set host off-line state
9105	gemdebugmode	EC	Binary	Bits enable individual AdeptGEM debug features.
9106	t1	EC	4-byte floating point	Receive timeout

Table 7-2
Predefined SVs, ECs and DVVALs (Continued)

ID	Reference Name	Class	Type	Description
9107	t22	EC	4-byte floating point	Protocol timeout
9108	t3	EC	4-byte floating point	Reply timeout
9109	t4	EC	4-byte floating point	Interblock timeout
9110	t5	EC	4-byte floating point	Connect separation timeout
9111	t6	EC	4-byte floating point	Control transaction timeout
9112	t7	EC	4-byte floating point	Connection idle timeout
9113	t8	EC	4-byte floating point	Network intercharacter timeout
9114	connecttime	EC	Unsigned 2-byte integer	Length of time the TCP/IP server connection is maintained
9115	retrylimit	EC	Unsigned 1-byte integer	Number of retries for failed block transmission
9116	duplblockdetect	EC	Boolean	TRUE = enable duplicate block detection
9117	convtimeout	EC	Unsigned 2-byte integer	Conversation timeout value (detects enquire/grant failures)
9118	remconfig9118	EC	Boolean	TRUE = allow operator to change process-related equipment constants in Remote state
9119	remconfig9119	EC	Boolean	TRUE = allow operator to change non-process related equipment constants in Remote state
9120	remconfig9120	EC	Boolean	TRUE = allow operator to initiate process program download in Remote state
9121	remconfig9121	EC	Boolean	TRUE = allow operator to select process program in Remote state
9122	remconfig9122	EC	Boolean	TRUE = allow operator to start process program in Remote state
9123	remconfig9123	EC	Boolean	TRUE = allow operator to pause/proceed process program in Remote state

Table 7-2
Predefined SVs, ECs and DVVALs (Continued)

ID	Reference Name	Class	Type	Description
9124	remconfig9124	EC	Boolean	TRUE = allow operator assist in Remote state
9125	remconfig9125	EC	Boolean	TRUE = allow operator to initiate material movement in Remote state (reserved for future use)
9126	remconfig9126	EC	Boolean	TRUE = allow operator to access menu items attached to a conditional section equivalent to the corresponding example on the GEM Control Panel menu page: ID 9126 = Conditional #50 ID 9127 = Conditional #51 ID 9128 = Conditional #52 ID 9129 = Conditional #53 ID 9130 = Conditional #54 ID 9131 = Conditional #55
9127	remconfig9127	EC	Boolean	
9128	remconfig9128	EC	Boolean	
9129	remconfig9129	EC	Boolean	
9130	remconfig9130	EC	Boolean	
9131	remconfig9131	EC	Boolean	
9132	montaskperiod	EC	4-byte floating point	Cycle rate for the limit monitoring task
9150	montasknum*	EC	Unsigned 1-byte integer	Trace Data Collection/Limits Monitoring task number (no longer used)
9151	equipid*	EC	Unsigned 2-byte integer	Equipment ID
9152	baud*	EC	Unsigned 2-byte integer	Baud rate for SECS-I
9153	commtasknum*	EC	Unsigned 1-byte integer	Communication channel task number (no longer used)
9154	commbuffsize*	EC	Unsigned 2-byte integer	Communications channel buffer size in KB
9155	servicetasknum*	EC	Unsigned 1-byte integer	GEM/SECS-II service task number (no longer used)
9200	alarmid	DV	Unsigned 2-byte integer	The ALID of alarm just set/cleared (user defined CEID)
9201	eventlimit	DV	List of integers	Limit ID crossed or list of IDs (user defined CEID)
9202	limitvariable	DV	Unsigned 2-byte integer	Variable ID of variable that changed zones (user defined CEID)
9203	ppchangename	DV	ASCII	Affected PPID (CEID 9300)

Table 7-2
Predefined SVs, ECs and DVVALs (Continued)

ID	Reference Name	Class	Type	Description
9204	ppchangestatus	DV	Unsigned 1-byte integer	Action taken on process program: 1 = created, 2 = edited, 3 = deleted (CEID 9300)
9205	transitiontype	DV	Binary	0 = low to high, 1 = high to low; zone transition type (user defined CEID)
9206	ecchanged	DV	Unsigned 2-byte integer	ID of EC that user changed (CEID 9200)
9207	ecnewvalue	DV	Same as type for EC that changed	New value of EC that changed (CEID 9200)
9208	ppvname	DV	ASCII	PPID verified and validated (CEID 9301)
9209	ppvstatus	DV	ASCII	Status of verification and validation (CEID 9301)
9210	lasttaskepschg	DV	Unsigned 1-byte integer	Task number of the most recent state change
9300	commdevname*	EC	ASCII	The name of the communication device. HSMS-SS: Name (or IP address) of the host and the port number. (e.g., 192,168,144,001 1) SECS-I: Serial port to use. (e.g., “SERIAL:1”).
9301	userdefsoftrev	EC	ASCII	If ECID 9102 is TRUE, the <SOFTREV> reported to the host will be the value of this ECID.
9302	ppmpath	EC	ASCII	The directory path used by all process program management messages.
(*) The host cannot modify the values of these Equipment Constants.				

7.2 The GEM Items Database

The GEM Items database contains information that defines GEM alarms, collection events, reports, and states. (The structure of this database is shown in **Table 8.2**) There are four record types in the GEM Items database: the Alarm record type, the Collection Event record type, the Report record type, and the State record type.

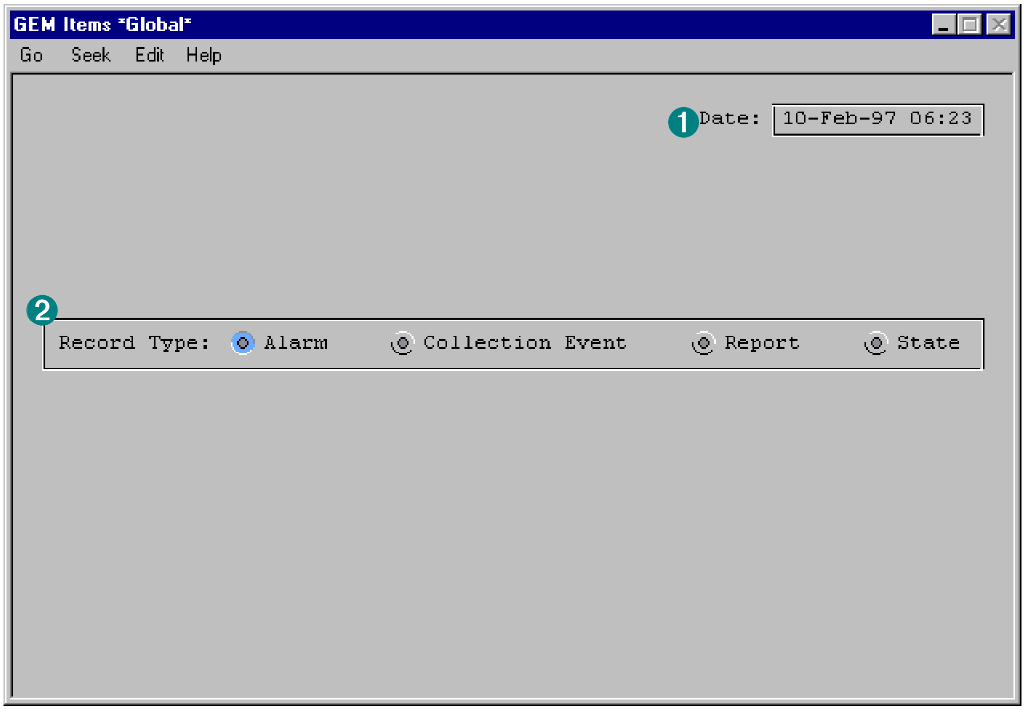
The GEM Items Menu Page

To GEM Items menu page is used to create records that define the GEM alarms, collection events, reports and states. These records are stored in the GEM Items database.

To create a new GEM Items record:

Edit ➡ GEM Items ➡ Edit ➡ New Record

The opening page is displayed:



Date: 10-Feb-97 06:23

Record Type: ● Alarm ○ Collection Event ○ Report ○ State

Figure 7-11
Initial GEM Items Menu Page

NOTE: The features of the page will change based on the record type selection (see item ❷.)

❶ Displays the date and time that the record was created.

❷ Choose a record type. The menu page changes based on the record type chosen.

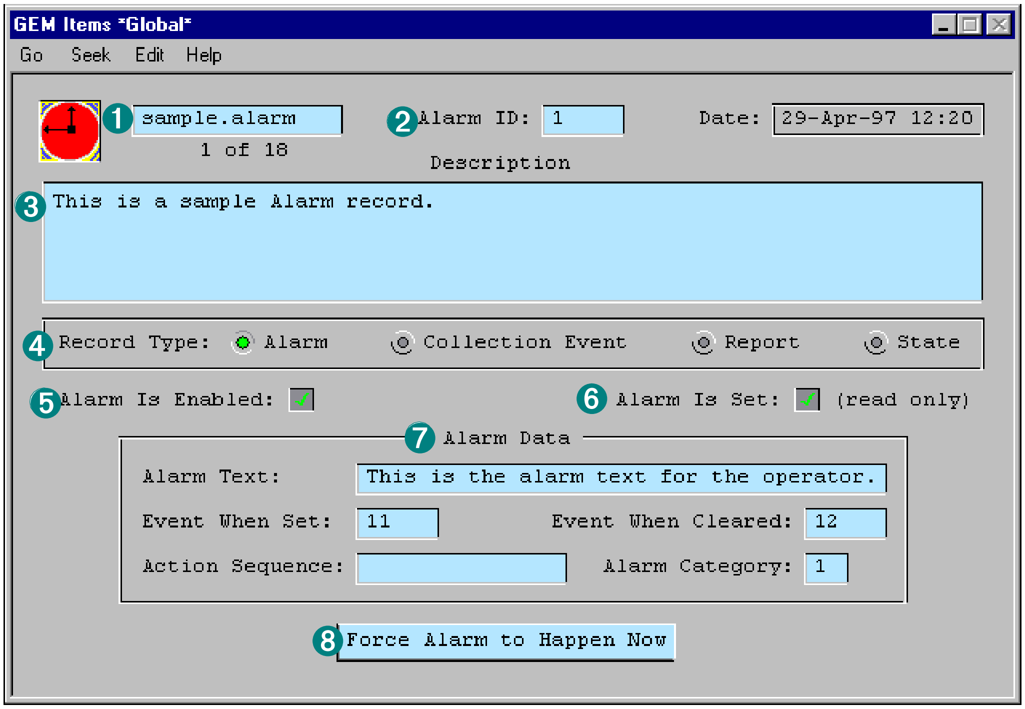
- Alarm: defines the alarms, their enabled status, and other parameters
- Collection Event: defines the Collection Events that are required by the GEM specification
- Report: stores the record ID numbers from the database and database record types that require information for a given report
- State: stores information on the state models used by GEM. This includes both the required state models such as the Control state model and variables for user-defined state models

Each record type is described in detail in the following sections.

The Alarm Record Type

The Alarm records define the alarms, their enabled status, and other parameters. These records are used to maintain the alarm monitoring that has been requested by the host. To open an Alarm record:

Edit ➡ GEM Items ➡ Seek ➡ Edit ➡ double-click record name



sample.alarm

1 of 18

Alarm ID: 1

Date: 29-Apr-97 12:20

Description

This is a sample Alarm record.

Record Type: ● Alarm ○ Collection Event ○ Report ○ State

Alarm Is Enabled: ☑

Alarm Is Set: ☑ (read only)

Alarm Data

Alarm Text: This is the alarm text for the operator.

Event When Set: 11

Event When Cleared: 12

Action Sequence:

Alarm Category: 1

Figure 7-12
GEM Items Database, Alarm Record

The Alarm ID for this example record is 1. The text “This is the alarm text for the operator.” will be used in all messages requiring alarm text. It shows the events that will be generated when the alarm is set or cleared. The enabled flag can be set by the host using the appropriate SECS-II message. You can also specify an AIM sequence to run when the alarm is set.

To define a new Alarm record:

1. Create a new GEM Items record.
2. Enter a name for the new record (item ❶).
3. Enter a unique Alarm ID number (item ❷).
4. Enter an optional description for this record (item ❸).
5. Select ● **Alarm** as the GEM Items record type (item ❹).
6. Select **Alarm is Enabled:** ☑ (item ❺) to enable the alarm. The **Alarm is Set:** ☑ checkbox (item ❻) is checked whenever the alarm is set (this is a read-only item).

Refer to item ❼ for steps 7 through 10:

7. Enter optional text that will be returned when an alarm is sent to the host.

8. Enter the numbers of the events (CEID) that will be generated when the alarm is set or cleared.

9. Enter the optional AIM sequence to run when this alarm is set. This sequence must be in the default control module. This resource module is normally MOWCTL, VWCTL, or PCBCTL unless the value of the default control module has been changed in the initialization database. See **“Alarm Management” on page 32** for details on how alarms are set.

10. Enter the optional Alarm Category number. This number will be reported to the host in the alarm code byte (ALCD).

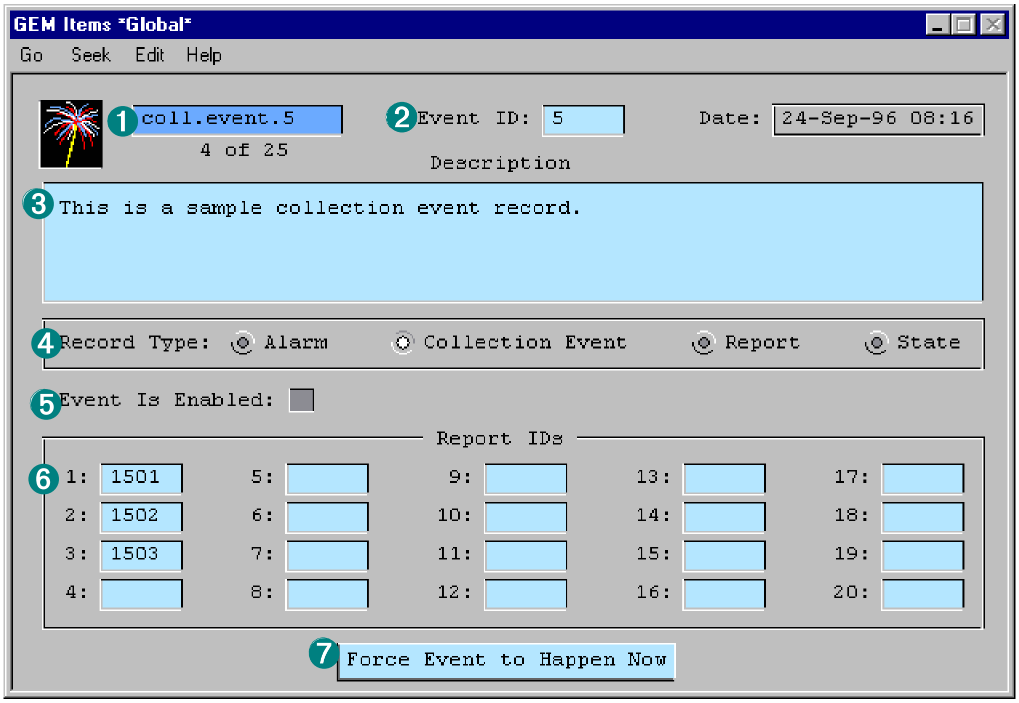
> **NOTE:** Although the alarm category is specified by the SECS-II standard, it is not required by the GEM standard. This value is supported by the AdeptGEM system for situations for which it is desirable to categorize alarms.

11. Choose **Force Alarm to Happen Now** (item ❽) to force the alarm (even if the alarm is not enabled). This is useful for testing the alarm setup.

The Collection Event Record Type

The Collection Event record type defines a Collection Events as required by the GEM specification. To access a Collection Event record:

Edit ➡ GEM Items ➡ Seek ➡ Edit ➡ double-click record name



coll.event.5

4 of 25

Event ID: 5

Date: 24-Sep-96 08:16

Description

This is a sample collection event record.

Record Type: ○ Alarm ○ Collection Event ○ Report ○ State

Event Is Enabled: □

Report IDs

1:	1501	5:		9:		13:		17:	
2:	1502	6:		10:		14:		18:	
3:	1503	7:		11:		15:		19:	
4:		8:		12:		16:		20:	

Force Event to Happen Now

Figure 7-13
GEM Items Database, Collection Event Record

The Collection Event ID number for this Collection Event record is 5. It has three Report records associated with it. When this collection event occurs, all the reports specified in the this record will be transmitted to the host. These reports are in turn made of a series of Variable IDs from the GEM Variables database.

To define a new Collection Event record:

1. Create a new GEM Items record.
2. Enter a name for the new record (item ❶).
3. Enter an ID number for this Collection Event (item ❷).
4. Enter an optional description for this record (item ❸).
5. Select ● **Collection Event** as the GEM Items record type (item ❹).
6. Select **Event is Enabled:** ☑ (item ❺) to enable this event.
7. Enter the Report ID numbers (item ❻)for the report records that will be transmitted when this Collection Event occurs.

   NOTE: Report IDs must be added to the collection event contiguously and starting with position number 1. The software will stop collecting reports when it finds the first 0 (or empty field) in the list.

8. Choose **Force Event to Happen Now** (item ❼)to force the collection event (even if the event is not enabled). This is useful for testing collection events.

Predefined Collection Event Records

The following table details the predefined Collection Event records:

Table 7-3
Predefined Collection Events

Event ID	Record Name	Description
9000	equipoffline	Control: Entry into OFF-LINE state
9001	controlstateloc	Control: Entry into LOCAL state
9002	controlstaterem	Control: Entry into REMOTE state
9003	controlstatechg	Control: 9000, 9001, or 9002 occurs
9004	operatorcmdiss	Control: Operator executes command while in the ON-LINE/REMOTE state
9100	processstart	Processing: AIM sequence started
9101	processcomplete	Processing: AIM sequence completed
9102	processstopped	Processing: AIM sequence stopped early (by the operator or host)
9103	procstatechange	Processing: Entry into any new processing state
9200	operatorconchg	Equipment Constants Capability: Operator changes constant locally

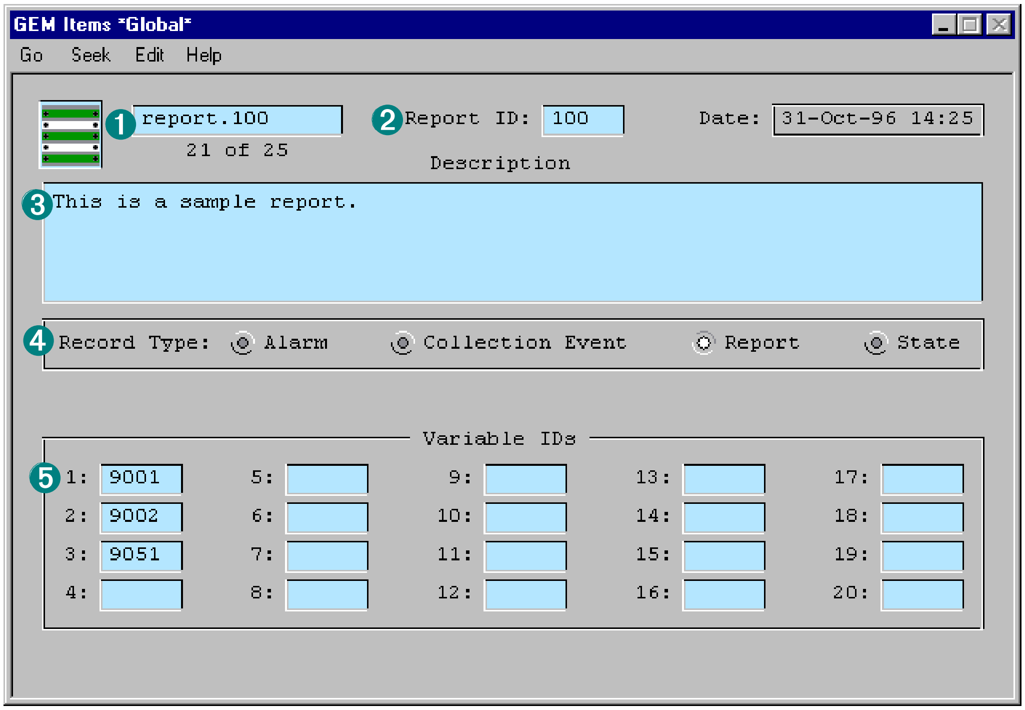
Table 7-3
Predefined Collection Events (Continued)

Event ID	Record Name	Description
9300	procprogchange	Process Program Management: AIM sequence created, deleted, or modified.
9301	procprogverval	Process Program Management: Program verified and validated
9400	procprogselect	AIM sequence selected for execution
9500	messagerecog	Terminal Services: Message acknowledged by operator.

The Report Record Type

The Report record type stores the VIDs from the GEM Variables database that make up a given report. These report definitions are used in the report capabilities such as variable data collection. To open a Report record:

Edit ➡ GEM Items ➡ Seek ➡ Edit ➡ double-click record name.



report.100

21 of 25

Report ID: 100

Date: 31-Oct-96 14:25

Description

This is a sample report.

Record Type: ○ Alarm ○ Collection Event ● Report ○ State

Variable IDs

1: 9001	5:	9:	13:	17:
2: 9002	6:	10:	14:	18:
3: 9051	7:	11:	15:	19:
4:	8:	12:	16:	20:

Figure 7-14
GEM Items Database, Report Record

This example record shows a report record made up of three Variable IDs: 9000, 9002, and 9151. These Variable IDs are defined in the GEM Variables database. When a report needs to be generated, AdeptGEM will search the appropriate database and automatically build a report in the proper SECS-II message format for delivery to the host.

To define a new Report record:

1. Create a new GEM Items record.
2. Enter a name for the new record (item ❶).
3. Enter a unique ID number for this Report record (item ❷).
4. Enter an optional description for this record (item ❸).
5. Select ◉ **Report** as the GEM Items record type (item ❹).
6. Enter the Variable ID numbers (item ❺) for the variables that will be collected when this report is generated.

> **NOTE:** Variable IDs must be added to the record contiguously and starting with position number 1. The software will stop collecting variables when it finds the first 0 (or empty field) in the list.

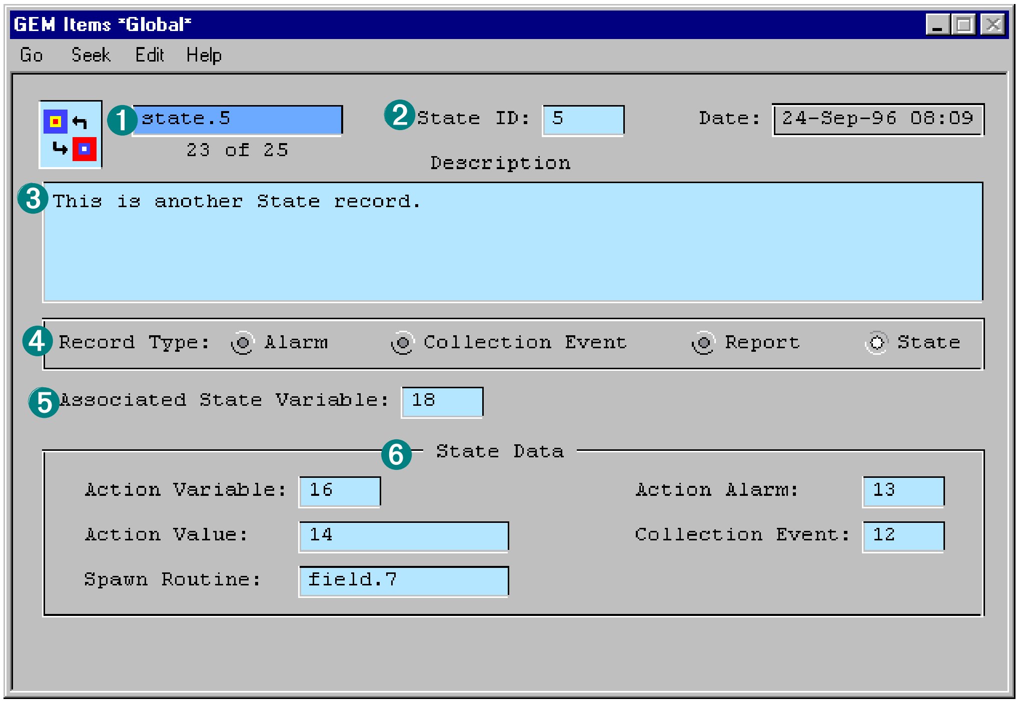
In general, the host will request the reports it is interested in. The host can request predefined reports or it can create and link its own reports. You need only define any predefined reports that the host will request.

When the host generates and links a report, AdeptGEM will create the report and give it a random name. This process is handled automatically by the system. If you view the report record type you may notice new report records with random generated record names. These are reports created at the request of the host.

The State Record Type

The State record type stores information on the state models used by GEM. To open the State Model record type:

Edit ➡ GEM Items ➡ Seek ➡ Edit ➡ double-click record name



state.5

23 of 25

State ID: 5

Date: 24-Sep-96 08:09

Description

This is another State record.

Record Type: ○ Alarm ○ Collection Event ○ Report ● State

Associated State Variable: 18

State Data

Action Variable: 16

Action Alarm: 13

Action Value: 14

Collection Event: 12

Spawn Routine: field.7

Figure 7-15
GEM Items Database, The State Record

Options on this page allow you to associate a state variable with a Variable ID from the GEM Variables database, specify an Alarm record associated with transition to this state, and specify a Collection Event record associated with transition to this state. You can also specify a V^+ routine to run on transition to this state.

To define a new State record:

1. Create a new GEM Items record.
2. Enter a name for the new record (item ❶).
3. Enter a unique identifier (item ❷) for a given state within a state model defined by the “Associated State Variable” (item ❺). (There cannot be duplicate State IDs in the database, even if

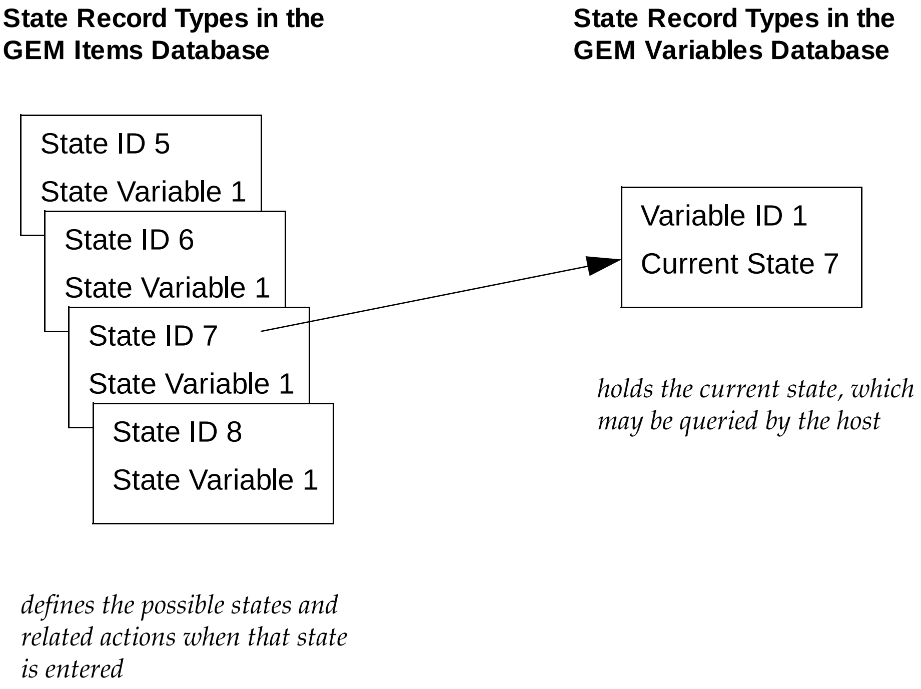
the Associated State Variables are different.) Normally, a given state model will have several different State records to describe the possible states.

4. Enter an optional description for this record (item ❸).
5. Select ◉ **State** as the GEM Items record type (item ❹).
6. Enter a unique number (item ❺) for all records associated with a given state model. (Must be the same number as a state variable data type record in the GEM Variables database.) This number creates a specific state model.

Refer to item ❻ for the following steps:

7. Enter the Variable ID of a record in the GEM Variables database. The Variable ID record specified here will be given the value specified in “Action Value”. This field can be left blank. (Note that this is not the same record as the “Associated State Variable”.)
8. Enter the value to give the GEM Variables database record specified in “Action Variable” when this state transition occurs.
9. Enter the name of the V^+ routine to run when this state is entered. See the routine **user.state.spawn( )** on page 209 for the calling parameters for this routine.
10. Enter the Alarm ID to set when entering the state.
11. Enter the Collection Event ID to trigger when this state is entered.

The following example shows the four states (State ID 5, 6, 7 & 8) associated with state model 1.



State Record Types in the GEM Items Database

State Record Types in the GEM Variables Database

holds the current state, which may be queried by the host

defines the possible states and related actions when that state is entered

Chapter 8
The AdeptGEM Database Structures

This chapter provides reference information on the structure of each database included in the AdeptGEM module. The databases specific to this module are:

- The GEM Variables Database
- The GEM Items Database

8.1 Structure of the GEM Variables Database

The GEM Variables database contains all of the GEM variables. The structure of this database is shown in **Table 8-1**

Table 8-1
Structure of the GEM Variables Database

Field Number **Variable**	**Field Name**	**Type** **Size [Array]**	**Description**
0 cc.name	Name	name 15	Name of variable or constant
1 cc.update	Update Date	date/time 4	The date/time this record was created or changed.
2 gm.f.desc	Description	string 72 [4]	Description of the variable. gm.asize.desc=4 Number of description lines
3 gm.f.menu= cc.page.name	Menu page name	name 15 [2]	Name of the menu page (and file) used to display the records. (Not currently used.)
4 gm.f.rec.type	Record Type	byte 1	Code indicating the type of variable. gm.rec.num=1 Record type for numeric variable gm.rec.str=2 Record type for string variable
5 gv.f.varbl.id	Variable ID	integer 2	ID number for the variable. This is the primary sort field.
6 gv.f.varbl.typ	Variable Type	byte 1	Specifies the AIM data type. See **Table 8-2**.

Table 8-1
Structure of the GEM Variables Database (Continued)

Field Number **Variable**	**Field Name**	**Type** **Size [Array]**	**Description**
7 gv.f.gemclass	GEM Data Class	byte 1	Specifies the GEM data type. See **Table 8-3**
8 gv.f.hostdtype	Host Data Type	byte 1	Data type used to send a value to host.
9 gv.f.prces.rel	Process Related	Boolean 1	Flag TRUE if the variable is related to process activity.
10 gv.f.acces.ctl	Access Control	byte 1	The units digit is compared to the current user access level to determine if changes are permitted; the tens digit controls access by the host.
11 gv.f.varbl.val	Value	real 4	Current value of integer/real constant, GEM/V$^+$ variable; signal number; **ai.ctl[ ]** index.
12 gv.f.min.value	Minimum Value	real 4	Minimum real value.
13 gv.f.max.value	Maximum Value	real 4	Maximum real value.
14 gv.f.typ.value	Typical Value	real 4	Typical real value.
15 gv.f.strng.val	String Value	string 72	The current value of the string variable.
16 gv.f.unit.idnt= gv.f.typ.strng	Units Ident/ Typical String	string 60	Numeric variable: SECS-II units identifier for the units of the variable. String variable: Typical value for string variable.
17 gv.f.lmt.event	Limit Monitor Event	integer 2	ID of the collection event associated with limit monitoring of this variable.

Notes:

- The “Variable ID” values must be unique throughout the database.
- All code that accesses the database must consider the record type.
- The database must be kept sorted.

Table 8-2
Values For Variable Type Field

Name	Value	Description
gv.typ.isig	1	Input signal (SV)
gv.typ.osig	2	Output signal (SV)
gv.typ.numv	4	Numeric variable (SV/EC)*
gv.typ.vpls	5	V^+ variable (SV)
gv.typ.ctlv	6	**ai.ctl[ ]** value (SV)
gv.typ.numf	51	Numeric function (SV/DVVAL)
gv.typ.strv	100	String variable (SV/EC)
gv.typ.strf	101	String function (SV/DVVAL)
(*) A numeric variable can also be a State variable.		

Table 8-3
Values For GEM Data Class Field

Name	Value	Description
gv.cls.sttvar	1	Data class for Status Variable*
gv.cls.eqpcon	2	Data class for Equipment Constant*
gv.cls.datval	3	Data class for Data Value*
gv.cls.statev	4	Data class for State Variable
(*) Value for data class must be a *bit number* in the mask used by the code.		

8.2 Structure of the GEM Items Database

Fields 1 through 4 are common to all record type definitions within the Items database. Refer to the specific record type for fields 5 through 15.

Table 8-4
Structure of the GEM Items Database

Field Number Variable	Field Name	Type Size [Array]	Description
0 cc.name	Name	name 15	Name of variable or constant
1 cc.update	Update Date	date/time 4	The date/time this record was created or changed.
2 gm.f.desc	Description	string 72 [4]	Description of the item. gm.asize.desc=4 Number of description lines
3 gm.f.menu= cc.page.name	Menu page name	name 15 [2]	Name of the menu page (and file) used to display the item records. (Not currently used.)
4 gm.f.rec.type	Record Type	byte 1	Code indicating the type of GEM item (alarm, collection event, report, or state) that is described by the record. (See Table 8-5.) This is the primary sort field.
5	ID	integer 2	
6	String	string 40	
7	Spawn Name	name 15	
8	Enable Flag	Boolean 1	
9	Set Flag	Boolean 1	
10	Integer 1	integer 2	
11	Integer 2	integer 2	
12	Integer 3	integer 2	
13	Integer 4	integer 2	
14	Real Value	real 4	

Table 8-4
Structure of the GEM Items Database (Continued)

Field Number Variable	Field Name	Type Size [Array]	Description
15	Integer Array	integer 2	

Notes:

- For fields 5 to 15, the field-number variables and the descriptions depend on the record type. See Tables 8-6, 8-7, 8-8, and 8-9.
- The routine **gm.next.rec( )** assumes that the "Record Type" field is the primary sort field.
- Any code that accesses the database must consider the record type.
- The database must be kept sorted, so that all the records of each type are kept together.
- The secondary sort on "Integer 1" is intended to group the State records for each State Model.

Table 8-5
Values For Record Type Field

Name	Value	Description
gm.rec.ala	1	Record type for Alarm records.
gm.rec.col	2	Record type for Collection Event records.
gm.rec.rep	3	Record type for Report records.
gm.rec.sta	4	Record type for State records.

Structure of the Alarm Record Type

The following table details the structure of the Alarm record type:

Table 8-6
Structure of the Alarm Record Type

Field Number Variable	Field Name	Description
5 gm.f.item.id	Alarm ID	ID number for the alarm (must be unique among all alarms).
6 gm.f.al.text	Alarm Text	Text sent to host for the alarm.
7 gm.f.spawn	Action Sequence	AIM sequence to run when the alarm is set.

Table 8-6
Structure of the Alarm Record Type (Continued)

Field Number Variable	Field Name	Description
8 gm.f.enabled	Alarm Enabled	This flag is set to TRUE when the alarm is enabled.
9 gm.f.al.set	Alarm Set	This flag is set to TRUE when the alarm is set.
10		Not used.
11 gm.f.al.setevnt	Set Event	ID of the triggered collection event triggered when the alarm is set.
12 gm.f.al.clrevnt	Clear Event	ID of the collection event triggered when the alarm is cleared.
13 gm.f.al.categor	Category	Alarm category reported to the host in the alarm code byte (ALCD). See the related note on **page 109**.
14 gm.f.al.action	Alarm Action	Code for the action to be performed in response to the alarm being set:* 0 = No action 1 = Pause sequence 2 = Abort sequence (*) This action is currently not performed.
15		Not used.
Note: • Fields 1 through 4 are common for all GEM Items record types. See **Table 8-4**		

Structure of the Collection Event Record Type

The following table details the structure of the Collection Event record type:

Table 8-7
Structure of the Collection Event Record Type

Field Number Variable	Field Name	Description
5 gm.f.item.id	Collection Event ID	ID number for the collection event (must be unique among all collection events).
6		Not used.
7		Not used.
8 gm.f.enabled	Collection Event Enabled	This flag is set to TRUE when the collection event is enabled.
9 to 14		Not used.

Table 8-7
Structure of the Collection Event Record Type (Continued)

Field Number Variable	Field Name	Description
15 gm.f.data.ids	Report IDs	IDs of reports associated with this collection event. gm.asize.cerpts=20 Number of reports.
Note: • Fields 1 through 4 are common for all GEM Items record types. See **Table 8-4**.		

Structure of the Report Record Type

The following table details the structure of the Report record type:

Table 8-8
Structure of the Report Record Type

Field Number Variable	Field Name	Description
5 gm.f.item.id	Report ID	ID number for the report (must be unique among all reports).
6 to 14		Not used.
15 gm.f.data.ids	Report IDs	IDs of variables listed by this report. (Set in the routine **gm.db.names( )** in the file GEMMOD.OVR.) gm.asize.rptvid=20 Number of Variable IDs (VIDs).
Note: • Fields 1 through 4 are common for all GEM Items record types. See **Table 8-4**.		

Structure of the State Record Type

The following table details the structure of the State record type:

Table 8-9
Structure of the State Record Type

Field Number Variable	Field Name	Description
5 gm.f.item.id	State ID	ID number for the state (must be unique among all the state records).
6		Not used.
7 gm.f.spawn	Action Spawn Routine	Name of the V^+ routine to be executed when the state is entered.

Table 8-9
Structure of the State Record Type (Continued)

Field Number **Variable**	**Field Name**	**Description**
8		Not used.
9		Not used.
10 gm.f.stt.varbl	State Variable	Variable ID for the State variable associated with the state. This is used as a secondary sort field so that all records for a Control model will be grouped together.
11 gm.f.stt.actvar	Action Variable	Variable ID for the variable to change upon entry to the state.
12 gm.f.stt.actce	Action Collection Event	ID of the Collection Event performed when the state is entered.
13 gm.f.stt.actalm	Action Alarm	ID of alarm set when the state is entered.
14 gm.f.stt.actval	Action Value	Value to assign to the action variable.
15		Not used.
Note: • Fields 1 through 4 are common for all GEM Items record types. See **Table 8-4**.		

Chapter 9 Descriptions of Routines in the AdeptGEM Module

This chapter describes the functions and calling sequences of routines contained in the AdeptGEM module. These routines may be called by application software written by a system customizer.

> **CAUTION:** The routines described in this section must be used whenever you access the GEM Variables or GEM Items databases. These routines invoke special locks that prevent the host and the equipment from simultaneously accessing the same database. These locks are not present in the normal AIM database routines and unpredictable results may occur if they are used in place of the AdeptGEM routines.

Each routine is presented on a separate page, in alphabetical order. The “dictionary page” for each routine contains the following sections, as applicable.

Calling Sequence

The format of a V$^+$ CALL instruction for the routine is shown.

> **NOTE:** The variable names used for the routine parameters are for explanation purposes only. Your application program can use any variable names you want when calling the routine.
>
> **NOTE:** Some calling sequences will not fit on a single line and are shown on two lines. However, all calling sequences must be entered on a single line in V$^+$ programs.

Function

This is a brief statement of the function of the routine.

Usage Considerations

This section is used to point out any special considerations associated with use of the routine.

Statement Syntax

For statement routines, this section shows the statement syntax.

Input Parameters

Each of the input parameters in the calling sequence is described in detail. For parameters that have a restriction on their acceptable values, the restriction is specified.

Output Parameters

Each of the output parameters in the calling sequence is described in detail.

Details

A complete description of the routine and its use is given.

File: The name of the program file in which this routine is contained.

Statement DB: For statement routines, the name of the statement database this routine references.

Related Routines

Other AIM routines, which are related to the function of the current routine, are listed.

> **NOTE:** Some of the routines listed may be documented in the reference guide for a different portion of your AIM system.

Calling Sequence

```
CALL disable_comm (args[], error)
```

Function

Statement execution routine for the DISABLE_COMM statement. It is used to perform the action of switching from the ENABLED to the DISABLED state in the Communications State model.

Usage Considerations

□ **Runtime** □ **Control** □ **Robot** □ **Vision**

Statement Syntax

```
DISABLE_COMM
```

Input Parameters

args[ ]	No arguments are used.

Output Parameters

error	Real variable that receives a value indicating whether or not the operation was successful and what action should be taken by the calling routine. 0 indicates the operation was successful (also returned if the model is *already* disabled). Otherwise, a standard AIM operator error response code is returned. See the standard AIM operator error response code values for details.

Details

This routine will disable host communication and place the equipment in the "not communicating" state.

Before communication can be successfully disabled, the communication channel must be properly installed and configured and communications with the host must be established. All the configuration options for the serial or ethernet channel as well as the GEM parameters are set in the GEM Variables database.

The equipment constants that define the communication configuration and GEM parameters are included in **Table 7-2**. They are also summarized in Tables **4-1** through **4-3**.

File: GEMSTMT.SQU

Statement DB: STATGEM.DB

Related Routines

Calling Sequence

```
CALL generate_alarm (args[], error)
```

Function

Statement execution routine for the GENERATE_ALARM statement. It is used to generate an alarm.

> **NOTE:** The statement name is hard-coded in the routine **gm.scr.spawn( )** (in the file GEMAIM.V2). That routine shows only alarms in the scrolling pick list.

Usage Considerations

□ **Runtime** □ **Control** □ **Robot** □ **Vision**

Statement Syntax

```
GENERATE_ALARM --gemitem--
```

Input Parameters

args[ ] Real array containing the arguments for this statement (record numbers or constants). The individual elements are described below:

`--gemitem--` `args[1]`
Record number of a record of type Alarm from the GEM Items database.

Output Parameters

error Real variable that receives a value indicating whether or not the operation was successful and what action should be taken by the calling routine. See the standard AIM operator error response code values for details.

Details

This statement sends an alarm message to the host. When this routine is executing, it temporarily locks access to the GEM Items database.

Refer to **section 3.8 on page 32** for details on alarm management. Refer to **page 119** for details on the structure of the Alarm record type.

File: GEMSTMT.SQU

Statement DB: STATGEM.DB

Related Routines

Calling Sequence

```
CALL generate_event (args[], error)
```

Function

Statement execution routine for the GENERATE_EVENT statement. It is used to generate a collection event.

> **NOTE:** The statement name is hard-coded in the routine **gm.scr.spawn( )** (in the file GEMAIM.V2). That routine shows only collection events in the scrolling pick list.

Usage Considerations

□ **Runtime** □ **Control** □ **Robot** □ **Vision**

Statement Syntax

```
GENERATE_EVENT --gemitem--
```

Input Parameters

args[ ] Real array containing the arguments for this statement (record numbers or constants). The individual elements are described below:

`--gemitem--` `args[1]`
Record number of a record of type Collection Event from the GEM Items database.

Output Parameters

error Real variable that receives a value indicating whether or not the operation was successful and what action should be taken by the calling routine. See the standard AIM operator error response code values for details.

Details

This statement sends a collection event to the host. When this routine is executing, it temporarily locks access to the GEM Items database.

Refer to **page 120** for details on the structure of the Collection Event record type.

File: GEMSTMT.SQU

Statement DB: STATGEM.DB

Related Routines

Calling Sequence

```
CALL gm.alarm (alid, set)
```

Function

Toggles an alarm ON or OFF, generates any “actions” associated with the alarm, and, if necessary, causes an alarm report message to be sent to the host.

Input Parameters

alid	The alarm identification number.
set	TRUE if the alarm should be turned on; otherwise, FALSE.

Details

This routine automatically handles the complexities associated with alarm management, such as checking to see if the host is on-line, changing or checking flags in an Alarm database record, generating events, calling spawn routines, etc.

If a message should be sent, it is not sent right away, but instead it is frozen (queued) to be sent later, thus allowing the calling program to continue without delay.

File: GEMENG.SQU

Related Routines

gm.ed.alarm
gm.get.alarm
gm.raw.alarm
gm.sr.alarm

Calling Sequence

```
CALL gm.com.test (status)
```

Function

Send an S1F13 message to see if the host is responding.

Output Parameter

status	0 if the host responds to the request; otherwise, a V+ error code.

Details

This routine sends an S1F13 message to the host. If the host responds as expected with an S1F14 response, **status** is set to 0.

File: GEMENGC.SQU

Related Routines

gm.estcom
gm.lineset

Calling Sequence

```
CALL gm.ed.alarm (alid, enable, status)
```

Function

Access an Alarm record in the GEM Items database to enable or disable the reporting of the selected alarm.

Input Parameters

alid The identification number of the alarm to have reporting enabled or disabled.

enable TRUE if the alarm should be enabled; otherwise, FALSE.

Output Parameter

status	0	Success
	1	There is no Alarm record with the given ALID
	< 0	Standard AIM database status return code

File: GEMDBASE.SQU

Related Routines

gm.alarm
gm.get.alarm
gm.raw.alarm
gm.sr.alarm

Calling Sequence

```
CALL gm.ed.event (ceid, enable, status)
```

Function

Access the Collection Event record in the GEM Items database to change the reporting status of a particular collection event. If a collection event is disabled, its associated reports (if any) will not be sent to the host.

Input Parameters

ceid	The identification number for the collection event to update.
enable	TRUE if the event should be enabled; otherwise, FALSE.

Output Parameter

status	0	Success
	1	There is no Collection Event record with the given CEID
	< 0	Standard AIM database status return code

File: GEMDBASE.SQU

Related Routines

gm.event
gm.get.event
gm.raw.event

Calling Sequence

```
CALL gm.eps.get (task, prev.state, eps.state)
```

Function

This subroutine returns the current or previous state of the main equipment processing state model.

Input Parameters

task	Task number for the task of interest (i.e., not the task index).
prev.state	Boolean TRUE to request the previous state, or FALSE to request the current state.

Output Parameter

eps.state The previous or current state, one of:

–1	Unknown
0	Running
1	Teach
2	Pause
3	Idle

Details

See the standard AIM routine **cu.set.mode( )** for more information.

File: GEMENG.SQU

Related Routines

cu.set.mode
gm.get.sinfo
gm.get.state

Calling Sequence

```
CALL gm.estcom (turn.on, status)
```

Function

Toggles the DISABLED/ ENABLED state in the Communications state model.

Usage Considerations

All configuration data must have legal values for an online request to succeed.

Input Parameter

turn.on	TRUE if communications should be enabled; otherwise, FALSE.

Output Parameter

status	0 if successful, otherwise a standard V$^+$ error code. Success is returned if the model is ALREADY in the desired state.

Details

AdeptGEM cannot go online if there is already another task running in the user-specified service task number, unless it is a dead SERVICE task, in which case we will try to abort it. This routine is similar to the Enable/Disable buttons on the control panels.

File: GEMENG.SQU

Related Routines

gm.com.test
gm.freeze.msg
gm.send.hostmsg
gm.usersend

Calling Sequence

```
CALL gm.event (ceid, force)
```

Function

Causes a collection event to be sent to the host (if appropriate) for the specified collection event identification number.

This subroutine will automatically check to see if the host is online, check to see if the collection event is actually enabled, fetch and pack the appropriate variables for each report (if any), etc. If a message should be sent, it is not sent right away, but instead it is frozen (queued) to be sent later, thus allowing the calling program to continue without delay.

Input Parameter

ceid The collection event identification number.

force (Optional) Boolean TRUE to cause the event to be sent even if it is not enabled. (Default is FALSE.)

File: GEMENG.SQU

Related Routines

gm.ed.event
gm.get.event
gm.raw.event

Calling Sequence

```
CALL gm.exist.vid (vid, type, exist)
```

Function

Check to see whether a given variable identification number exists.

Input Parameters

vid The identification number to test.

type Bit mask indicating the type of variable to check for:
Bit 1 Status variable
Bit 2 Equipment constant
Bit 3 Data value

Output Parameter

exist TRUE if the vid was found and its type (SV, EC or DVVAL) was one of the types being checked for; otherwise, FALSE.

Details

Status variables, equipment constants or data values can be exclusively searched for, or any combination of the three classes can be searched.

If there is an unusual error, such as an error in accessing one of the databases, then this subroutine will assume that the VID does not exist.

File: GEMENG.SQU

Related Routines

gm.get.vid.list
gm.get.vinfo

Calling Sequence

```
CALL gm.freeze.msg (reply, hd[], $data[], end.marker, status)
```

Function

This subroutine provides an alternative to **gm.usersend( )**.

Instead of sending a message to the host immediately and waiting (an arbitrary period of time) for a reply (if any) this routine allows you to “freeze”, or queue, a message, which will be sent later by the service task.

The calling program can then continue without delay.

Input Parameters

reply TRUE if a reply is desired; otherwise, FALSE.

hd[ ] An array representing the header for the message, with the **uc.hd.stream**, **uc.hd.function** and **uc.hd.sys.#** elements filled. The routine **gm.new.head( )** is the most appropriate way to create this header.

$data[ ] The stream of bytes, fully PACK’ed, representing the message. “**s2.pack.***” subprograms can be used to create this $data[ ] array.

end.marker The total length, in bytes, of the message. Or, equivalently, the byte number (starting at 1) of the first byte in the data stream to NOT send; hence an “end” location.

Output Parameter

status 0 if the message was successfully frozen, or 1 if not (meaning that the queue is FULL).

Details

The size of the queue is controlled by the global variable **gm.fz.size**, which defaults to 10. The size specified is a count of the number of messages that can be held, not the total message size.

File: GEMENG.SQU

Related Routines

gm.com.test
gm.estcom
gm.new.head
gm.usersend

Calling Sequence

```
CALL gm.get.ainfo (alid, event[], act, $spawn, status)
```

Function

Access the Alarm records in the GEM Items database to retrieve action information for a specified alarm: collection events to use, what action to take, and the name of a spawn routine to call.

Input Parameter

alid The identification number for the desired alarm.

Output Parameters

event[ ] An array with the alarm clear collection event identification number in element 0 and the alarm set CEID in element 1.

act This parameter is not currently used.

$spawn The name of an AIM sequence to execute if the alarm is set. If none, an empty string.

status
0 Success
1 ALID does not exist
< 0 Standard AIM database status return code

Details

File: GEMDBASE.SQU

Related Routines

gm.alarm
gm.ed.alarm
gm.get.alarm

Calling Sequence

```
CALL gm.get.alarm (alid, alcd, $altx, enabled, status)
```

Function

Access the Alarm record in the GEM Items database to retrieve information about an alarm: a brief description text, whether the reporting of the alarm to the host is enabled or not, and whether the alarm is set or not.

Input Parameter

alid The identification number for the desired alarm (ALID).

Output Parameters

alcd Alarm code byte (ALCD)

Bit 8	(mask ^H80)	Set if alarm is set
Bit 7-1	(mask ^H7F)	Alarm category

NOTE: The GEM standard does not use the alarm category, which is supported by the SECS-II standard. The alarm category is supported by the AdeptGEM system for situations for which it is desirable to categorize alarms.

$altx The alarm text string (ALTX), 40 characters maximum.

enabled TRUE if the reporting of the alarm to the host is enabled; otherwise, FALSE.

status

0	Success
1	ALID does not exist
< 0	Standard AIM database status return code

Details

File: GEMDBASE.SQU

Related Routines

gm.alarm
gm.ed.alarm
gm.raw.alarm
gm.sr.alarm

Calling Sequence

```
CALL gm.get.clock (secs.ii, $clock)
```

Function

Return the value of the AdeptGEM internal clock.

Input Parameter

secs.ii TRUE if a SECS-II format string is desired; otherwise, FALSE if the traditional Adept format is desired. This parameter can be omitted (FALSE is assumed).

Output Parameter

$clock A string in the form “dd-MMM-yy hh:mm:ss” (Adept format) or “yyyymmddhhmmsscc” (SECS-II format).

Details

This routine can be used for GEM time-stamping, which must be accurate to the nearest hundredth of a second (centisecond). This function returns a string in either the precise SECS-II format or the traditional Adept format.

File: GEMENG.SQU

Calling Sequence

```
CALL gm.get.event (ceid, rptid[], num.ids, enabled, status)
```

Function

Access the Collection Event records in the GEM Items database and retrieve information regarding an event: a list of associated reports, and whether or not reporting of the event to the host is enabled.

Input Parameter

ceid The identification number of the desired collection event.

Output Parameters

rptid[ ] An array containing the RPTIDs, starting at element zero.

num.ids The total number of RPTIDs in the array **rptid[ ]**.

enabled TRUE if reporting is enabled; otherwise, FALSE.

status:

0	Success
1	There is no Collection Event record with the given CEID
< 0	Standard AIM database status return code

Details

File: GEMDBASE.SQU

Related Routines

gm.ed.event
gm.event
gm.raw.event

Calling Sequence

```
CALL gm.get.report (rptid, vid[], num.ids, status)
```

Function

Access the Report records in the GEM Items database and retrieve a list of variable identification numbers associated with a specified report.

Input Parameter

rptid	The identification number of the desired report.

Output Parameters

vid[ ]	An array containing the VIDs, starting at element zero.
num.ids	The total number of VIDs.
status	0 Success 1 RPTID does not exist < 0 Standard AIM database status return code

Details

File: GEMDBASE.SQU

Related Routines

gm.make.links
gm.make.report

Calling Sequence

```
CALL gm.get.sinfo (vid, state, $spawn, id[], act.val, status)
```

Function

Access the State records in the GEM Items database to obtain various information about a particular state associated with a state variable. This includes the name of a spawn routine to execute upon a transition into the state, a variable to change the value of and its new value, a collection event to activate and an alarm to set.

Input Parameters

vid The identification number of the state variable associated with the desired state.

state The ID number for the desired State record.

Output Parameters

$spawn The name of a V^+ routine to execute. If none, an empty string.

id[ ] An array of identification numbers:
0 Action variable (0 if none)
1 Collection event (0 if none)
2 Alarm to set (0 if none)

act.val The new value for the “action variable”, if appropriate. (The value of the variable is not changed by this routine.)

status
0 Success
1 Specified State record does not exist in the GEM Items database
2 Specified State record does exist, but the associated VID specified in the record does not match the vid input parameter
< 0 Standard AIM database status return code

Details

File: GEMDBASE.SQU

Related Routines

gm.eps.get
gm.get.state

Calling Sequence

```
CALL gm.get.state (state, vid, $state, status)
```

Function

This routine accesses a State record in the GEM Items database to retrieve the name for a state associated with a particular state variable.

Input Parameters

state The identification number for the state of interest

vid The identification number of the state variable expected to be associated with the specified state

Output Parameters

$state The name of the state (1-15 characters)

status
0 Success
1 Specified State record does not exist in the GEM Items database
2 Specified State record does exist, but the associated VID specified in the record does not match the vid input parameter
< 0 Standard AIM database status return code

Details

File: GEMDBASE.SQU

Related Routines

gm.eps.get
gm.get.sinfo

Calling Sequence

```
CALL gm.get.str (strid, $str, vtype, gtype, status)
```

Function

Access the “string” database to retrieve a configuration string.

Input Parameter

strid	the identification number of the desired string.

Output Parameters

$str	the string requested.
vtype	Variable type code (returns 0 if error)
gtype	GEM variable class (returns 0 if error)
status	0 success 1 string identification number does not exist < 0 standard AIM database status return code

Details

File: GEMDBASE.SQU

Related Routines

gm.get.strinfo
gm.set.str

Calling Sequence

```
CALL gm.get.strinfo (vid, $name, pr, host.rw, $typical, status)
```

Function

Accesses a string variable in the GEM Variables database to retrieve miscellaneous information regarding the string: a name for the string, a flag indicating whether or not the string is process related and a typical (default) value for the string.

Input Parameter

vid	The identification number of the string variable for which the information is desired.

Output Parameters

$name	The name of the string (1-15 characters).
pr	TRUE if the string is process related; otherwise, FALSE.
host.rw	TRUE if the variable can be modified by the host.
$typical	The typical or default value.
status	0 Success 1 No string variable with the specified ID exists < 0 Standard AIM database status return code

Details

File: GEMDBASE.SQU

Related Routines

gm.get.str
gm.set.str

Calling Sequence

```
CALL gm.get.val (vid, value, status)
```

Function

Return the actual value of a numeric variable defined in the GEM Variables database.

Usage Considerations

This routine will not work for GEM Data Values (DVVALS).

Input Parameter

vid	The identification number of the variable whose value is to be retrieved.

Output Parameters

value	The value of the variable.	
status	0	Success
	1	Requested variable does not exist
	2	Requested variable exists, but its value could not be determined
	<0	Standard AIM database status return code

Details

This routine performs the equivalent of calling **gm.get.var( )** and then **gm.var2val( )**. Thus, for GEM real-value variables, the value in the database is returned. However, for **ai.ctl[ ]** values and V$^+$ variables, the value of the referenced variable is returned.

To obtain the value of a string variable, use **gm.get.str( )**.

File: GEMENG.SQU

Related Routines

gm.get.str
gm.get.var
gm.pack.val
gm.set.var
gm.var2val

Calling Sequence

```
CALL gm.get.var (vid, value, vtype, gtype, htype, status)
```

Function

Access the GEMVariable database to retrieve value information from the specified variable.

Input Parameter

vid The identification number of the variable for which value information is needed.

Output Parameters

value The raw “value” field in the variable record, which may indicate the variable’s value, a signal number, an **ai.ctl[ ]** index, etc., depending on the value of **vtype**.

vtype The main type of the variable:

1	Input signal
2	Output signal
3	Numeric constant
4	GEM variable
5	V^+ variable
6	**ai.ctl[ ]** value

gtype GEM variable class:

1	Status variable
2	Equipment Constant
3	Data value
4	State variable

htype The SECS-II type as seen by the host, such as ^52 for 2-byte unsigned integer.

status

0	Success
1	VID does not exist
< 0	Standard AIM database status return code

Details

File: GEMDBASE.SQU

Related Routines

gm.set.var
gm.var2val

Calling Sequence

```
CALL gm.get.vflags (vid, proc.rel, host.rw, status)
```

Function

Access the GEM Variables database to retrieve Boolean flag information associated with a variable.

Input Parameter

vid The identification number of the variable for which information is desired.

Output Parameters

proc.rel TRUE if the variable is process related; otherwise, FALSE.

host.rw TRUE if the variable can be modified by the host at all; otherwise, FALSE.

status

status	0	Success
	1	VID does not exist
	< 0	Standard AIM database status return code

Details

File: GEMDBASE.SQU

Calling Sequence

```
CALL gm.get.vid.list (mask, list[], num)
```

Function

Return a list of existing VIDs .

Input Parameter

mask A bit field specifying the class or classes of variables desired in the list, as follows:

Bit 1(^B1)	Status variables
Bit 2(^B10)	Equipment constants
Bit 3(^B100)	Data values

Output Parameters

list[ ] A real array containing the list of VIDs, starting with element 0.

num The total number of elements in **list[ ]**.

Details

The list can be of status variables only, equipment constants only, data values only, or any combination. The VIDs in the returned list are guaranteed to be in ascending order.

File: GEMENG.SQU

Related Routines

gm.exist.vid
gm.get.vinfo

Calling Sequence

```
CALL gm.get.vinfo (vid, $name, lmevent, range[], $units, status)
```

Function

Access the GEM Variables database to retrieve miscellaneous information associated with a variable.

Input Parameter

vid The identification number of the variable for which information is desired.

Output Parameters

$name The (up to) 15-character name of the variable.

lmevent The limits monitoring event ID number.

range[ ] Limit information in the following elements:
[0] = Typical (default) value
[1] = Minimum value
[2] = Maximum value

$units The units string for the variable.

status
0 Success
1 ID does not exist
< 0 Standard AIM database status return code

Details

File: GEMDBASE.SQU

Related Routines

gm.exist.vid
gm.get.vid.list

Calling Sequence

```
CALL gm.lineset (online)
```

Function

This subprogram implements the steps to set the equipment On-Line or Off-Line.

Usage Considerations

This subprogram should NOT be used if the host is requesting On- or Off-Line. It is meant only for the equivalent of operator control. The host On- or Off-Line situation is implemented in **gm.stream1( )**.

Input Parameter

online	TRUE if the equipment should be turned On-Line; otherwise, FALSE for Off-Line.

Details

Bringing the equipment On-Line involves a successful transaction with the host, so even after this subprogram returns, it may be a short while before the equipment actually goes On-Line or it may fail entirely. Global variables such as **gm.online** can be monitored to track the actual control state. This routine is similar to the Online/Offline buttons on the control panels.

File: GEMENGC.SQU

Related Routine

gm.streamN

Calling Sequence

```
CALL gm.log ($message, code)
```

Function

Log errors and other informative debug messages.

Usage Considerations

The action of this routine is controlled with equipment constant 9105 (**gemdebugmode**) in the GEM Variables database. Any of the output destinations can be individually enabled or disabled with that variable.

Input Parameters

$message	The message to write to the window (along with a time stamp and task number added by this routine).
code	Optional AIM error code. If this parameter is specified, the corresponding error message is added to the output.

Details

The messages processed by this routine have the format:

```
hh:mm:ss (task) <contents of '$message'><error string>
```

where

hh:mm:ss	is the current time
task	is the number of the current V+ task
<error string>	is the standard AIM error string corresponding to the value of the **code** parameter, or blank if **code** is not defined

The destination for the message is determined by the value of equipment constant 9105 (**gemdebugmode**), and by the number of the V+ task that calls this routine, as shown below.

Bit set in EC9105	Task Number	Destination of Message
2	gm.stask	GEM Service window
3	Not gm.stask	V+ Monitor window
4	Any	Record the message in a temporary log of recent messages, which is available to the host as status variable 9010

Notes: Bit #1 of EC9105 controls whether or not messages from the SECS-I/HSMS-SS service task are displayed in the “SECS-I Debug” or “HSMS-SS Debug” window. Output to those windows does not use this routine.

The number of messages retained in status variable 9010 is controlled by the V+

global variable **gm.temp.log2siz**, as shown below. The variable **gm.temp.log2siz** must have one of the values shown.

gm.temp.log2siz	=	0	1	2	3	4	5	6	7
Messages in log	=	1	2	4	8	16	32	64	128

File: GEMENG.SQU

Calling Sequence

```
CALL gm.make.links (ceid, rptid[], num, status)
```

Function

Accesses the Collection Event records in the GEM Items database to form links from a collection event to one or more reports.

Usage Considerations

A database record must already exist for the specified collection event, but it must not reference any RPTIDs.

Input Parameters

ceid The identification number for the desired collection event.

rptid[ ] An array containing one or more RPTIDs (links) in elements 0 through **num**–1.

num The total number of RPTIDs (links) in the array **rptid[ ]**.

Output Parameter

status

0	Success
1	There is no Collection Event record with the given CEID
2	One or more RPTIDs (links) are already defined for the specified collection event
< 0	Standard AIM database status return code

Details

This routine does *not* verify that the given RPTID numbers are valid (the subroutine **gm.get.report( )** can be used to determine if a report exists). If there already are one or more links defined, the old links are *not* destroyed and an error is returned.

File: GEMDBASE.SQU

Related Routines

gm.get.report
gm.make.report

Calling Sequence

```
CALL gm.make.report (rptid, $name, vid[], num, status)
```

Function

Access the GEM Items database and create a new Report record.

Usage Considerations

If a Report record already exists with the given report identification number, that record will NOT be modified.

Input Parameters

rptid	The identification number of the new report.
$name	The name for this record, up to 15 characters.
vid[ ]	VIDs for the variables to be reported by the report, in array elements 0 through **num**–1.
num	The number of VIDs in the array **vid[ ]**.

Output Parameter

status:	0	Success
	1	A Report record with the specified RPTID already exists
	< 0	Standard AIM database status return code

Details

The report record is created as follows:

Name	Set to the value of the “$name” parameter.
Report ID	Set to the value of the **rptid** parameter.
Update Date	Set to the current the date/time.
Description	NULL strings.
VIDs	Set array elements to the values in the **vid[ ]** parameter. **num** elements are set; the rest (if any) are set to zero in the record to indicate they are not used.

This routine does *not* verify that the VIDs are valid.

File: GEMDBASE.SQU

Related Routines

gm.get.report
gm.make.links

Calling Sequence

```
CALL gm.new.head (stream, function, hd[])
```

Function

Create the SECS-II message header necessary for **gm.usersend( )** and **gm.freeze.msg( )**. An array is created with the following elements filled: uc.hd.stream, uc.hd.function, uc.hd.sys.1, uc.hd.sys.2, uc.hd.sys.3 and uc.hd.sys.4.

Input Parameters

stream	The stream number to put in the header.
function	The function number to put in the header.

Output Parameter

hd[ ]	The created message header adequate for passing to the routine **gm.usersend( )**

Details

The stream and function elements are filled with the numbers specified as input arguments; the system bytes are filled with the next available message ID number and the current task number (and the global message ID number variable is incremented), thus fulfilling the uniqueness requirement of system bytes in SECS-II messages (as long as this subroutine is used consistently).

> **NOTE:** This routine should not be used for replies to host primary messages.

File: GEMENG.SQU

Related Routines

gm.freeze.msg
gm.usersend

Calling Sequence

```
CALL gm.pack.mdln ($data[], arrow)
```

Function

Pack the SECS-II data items <MDLN> and <SOFTREV> as a two-element list into a stream.

Input Parameters

$data[ ]	The $data[ ] array to receive the list.
arrow	The byte position in the $data[ ] array to start writing data to. The first byte in the $data[ ] array is number 1.

Output Parameters

$data[ ]	The modified $data[ ] array.
arrow	The byte position in the $data[ ] array just following the last byte packed.

Details

This routine is used to generate the data for messages S1F2, S1F13 and S1F14.

The Equipment Model Type, <MDLN>, is an ASCII string item that identifies the equipment to the host. This routine defines the value as “Annnnn”, where “nnnnn” is one to five digits representing the model number of the Adept MV controller, as supplied by the V^+ function ID(1,1).

The Software Revision Code, <SOFTREV>, is an ASCII string item that identifies the software running on the equipment. The string is defined either in the GEM Variable database by equipment constant 9301, or as a predefined string with the format “V+vv.r”. The equipment constant is used if (1) equipment constant 9102 is defined and has a TRUE (i.e., nonzero) value, and (2) equipment constant 9301 is defined. If those conditions are not satisfied, a string identifying the V^+ system version is used. In that case, “vv” is the V^+ version number supplied by the V^+ function ID(3,1), and “r” is the V^+ revision number supplied by the V^+ function ID(4,1).

File: GEMENG.SQU

Calling Sequence

```
CALL gm.pack.val (vid, $data[], arrow, status)
```

Function

Pack the value of a status variable, equipment constant or data value into a SECS-II message that is being built.

Input Parameters

vid	The variable identification number of the variable whose value is to be packed.
$data[ ]	The array of strings representing the packed contents of the SECS-II message that is being built.
arrow	The byte position in the $data[ ] array where the next item should be packed.

Output Parameters

$data[ ]	The $data[ ] array which now contains an additional data item if the pack was successful; otherwise, the unaltered $data[ ] array.
arrow	The position in the $data[ ] array immediately following the data item that was packed (or the most recently successfully packed data item, in case of error).
status	0 if success, or else a standard AIM error code.

Details

For variables not defined by Adept, this routine calls **gm.user.packval( )** to allow for custom processing. That provides the mechanism for custom status variables and data values.

> **NOTE:** If there is an error—such as an undefined VID or the value cannot be fetched—then this routine will not pack anything, and it is up to the calling program to decide what to do next (abort, pack a null item, etc.).

File: GEMENG.SQU

Related Routines

gm.get.str
gm.get.val
gm.get.var
gm.user.packval
gm.var2val

Calling Sequence

```
CALL gm.raw.alarm (lrec, alid, alcd, $altx, enabled, status)
```

Function

Access the Alarm records in the GEM Items database to retrieve information about the alarm in a specified logical record number.

Input Parameter

lrec The logical record number for the Alarm record to retrieve (1, 2, ...).

Output Parameters

alid The alarm identification number of the alarm found in the database (ALID).

alcd Alarm code byte (ALCD)

Bit 8	(mask ^H80)	Set if alarm is set
Bit 7-1	(mask ^H7F)	Alarm category

NOTE: The GEM standard does not use the alarm category, which is supported by the SECS-II standard. The alarm category is supported by the AdeptGEM system for situations for which it is desirable to categorize alarms.

$altx The (up to) 40-character alarm text string (ALTX).

enabled TRUE if the reporting of the alarm to the host is enabled; otherwise, FALSE.

status

0	Success
1	Database record does not exist
< 0	Standard AIM database status return code

Details

The most common use of this subroutine would be to retrieve the first record, then the second, then the third, etc., until status returns 1, which would achieve a full traversal of the Alarm records in the database.

Within the GEM Items database, the record type (Alarm, Event, Report, State) is used as the primary sort field. Thus, even though there are four distinct types of records included in the same database, the records are not “mixed up” and the database can be thought of as four separate databases connected end-to-end. The idea of logical records still applies within each group.

File: GEMDBASE.SQU

Related Routines

gm.alarm
gm.ed.alarm
gm.get.alarm
gm.sr.alarm

Calling Sequence

```
CALL gm.raw.event (lrec, ceid, rptid[], num.ids, enabled, status)
```

Function

Access the Collection Event records in the GEM Items database and retrieve information regarding the event in a specified logical record number in the database: a list of associated reports (links) and a flag indicating whether or not reporting of the event to the host is enabled.

Input Parameter

lrec The logical record number for the Collection Event record to retrieve (1, 2, ...).

Output Parameters

ceid The identification number of the collection event found in the database.

rptid[ ] An array containing the RPTIDs, starting at element zero.

num.ids The total number of RPTIDs returned in **rptid[ ]**.

enabled TRUE if reporting is enabled; otherwise FALSE.

status
0 Success
1 Database record does not exist
< 0 Standard AIM database status return code

Details

The most common use of this subroutine would be to retrieve the first record, then the second, then the third, etc., until status returns 1, which would achieve a full traversal of the Event records in the database.

Within the GEM Items database, the record type (Alarm, Event, Report, State) is used as the primary sort field. Thus, even though there are four distinct types of records included in the same database, the records are not “mixed up” and the database can be thought of as four separate databases connected end-to-end. The idea of logical records still applies within each group.

File: GEMDBASE.SQU

Related Routines

gm.ed.event
gm.event
gm.get.event

Calling Sequence

```
CALL gm.remote.cmd ($name, $argname[], $arg[], num,
                                            argstat[], status)
```

Function

This routine implements AIM functionality for the GEM Remote Control additional capability.

Input Parameter

$name

Name of the “remote” command to be done:
- One of the predefined GEM commands (PANIC or SPEED)
- A V^+ program name
- An AIM sequence

$argname[ ]

Argument names (elements 0 to num–1):
[0] = Name of argument 1
[1] = Name of argument 2

$arg[ ]

Argument values (elements 0 to num–1):
[0] = Value of argument 1
[1] = Value of argument 2

num

Number of elements in **$argname[ ]** and **$arg[ ]**

Output Parameter

argstat[ ]

Element N reports the status of input parameters **$argname[N]** and **$arg[N]**, as follows:
0 Both parameters are okay
1 One or both parameters does not exist
2 One or both parameters has an illegal value

status

Completion status of the request, as follows:
0 Okay
1 Command is not known
2 Cannot perform command right now
3 At least one parameter is invalid (i.e., at least one **argstat[ ]** <> 0)

Details

With a SECS-II S2F41 message, the host can request any of the following:

1. Execute one of these standard commands:

   PANIC (no arguments)

   SPEED VALUE speed_value TASK task_number

2. Spawn a V^+ routine

   If a V^+ routine is to be spawned, it is expected to have the same parameter list as this routine. See **gm.user.rc( )** on page 173 for more information.

3. Spawn an AIM sequence

If an AIM sequence is spawned, its argument values (but not their names) are passed in **$ai.ctl[gm.rc.args+i]** (i=0, 1, 2,..., num–1). The global variable **gm.rc.args** can be initialized or changed by custom code (for example, in the routine **gm.user.init**). If it is found undefined, it is set to 350.

> **NOTE:** The **$name** parameter is checked in the order above. Thus, there must not be any V$^+$ routine with the same name as or sequence spawn that might be requested.

File: GEMRCPPM.SQU

Related Routines

gm.user.rc
user.remote.cmd

Calling Sequence

```
CALL gm.send.hostmsg ($text[], start, num, status)
```

Function

Queue a message to be sent to the host. The SECS-II message used is S10F1, which allows for up to 237 characters.

Usage Considerations

The host must be communicating and on-line.

Input Parameters

$text[ ]	An array containing the message to send. If the message is stored in multiple strings, every string except the first must begin with $CHR(0).
start	The first element of **$text[ ]** to use (often 0, but not necessarily).
num	The total number of elements in **$text[ ]** to use.

Output Parameter

status

0 The message was successfully queued.
1 Could not queued (queue full)
2 Message too long
3 Host not on-line and communicating

Details

File: GEMENG.SQU

Calling Sequence

```
CALL gm.seq.spawn ($module, $sequence, status)
```

Function

The routine will execute an AIM control sequence in the current task.

Input Parameters

$module The module name

$sequence The sequence name

Output Parameter

status Status of the sequence spawn:

If the current task is the main menu task, any error results in an error popup and **status** is set to **rn.opr.abort**.

If the current task is any other task,

0 If successful

1 If sequence or module name is not defined

2 If sequence cannot be run

3 If unsuccessful because of other reasons (e.g., the sequence failed with an error)

Details

This routine executes the specified sequence in the current V$^+$ task. Thus, execution of the task is suspended until the sequence completes. For this reason, the sequence is required to be a control sequence, and it should complete execution as quickly as possible.

File: GEMAIM.SQU

Calling Sequence

```
CALL gm.set.str (vid, $string, status)
```

Function

Set the value of a string variable in the GEM Variables database.

Input Parameters

vid	The identification number of the string variable whose value is to be changed.
$string	The new value for the string (an empty string is allowed as the value).

Output Parameter

status	0	Success
	1	No string variable with the specified ID exists
	< 0	Standard AIM database status return code

Details

File: GEMDBASE.SQU

Related Routines

gm.get.str
gm.get.strinfo

Calling Sequence

```
CALL gm.set.var (vid, value, status)
```

Function

Access the GEM Variables database to change the Value field for a desired variable. Note that the Value field is not necessarily the actual value of the variable, but could be a signal number or **ai.ctl[ ]** index.

Input Parameters

vid	The identification number of the variable to change.
value	The new value for the Value field.

Output Parameter

status	0	Success
	1	Specified variable does not exist
	< 0	Standard AIM database status return code

Details

File: GEMDBASE.SQU

Related Routines

gm.get.val
gm.get.var
gm.set.str
gm.var2val

Calling Sequence

```
CALL gm.sr.alarm (alid, set, status)
```

Function

Accesses the Alarm records in the GEM Items database to set or reset a specified alarm.

Usage Considerations

This routine only changes the enable/disable flag, it does not trigger any of the events specified in the alarm record.

Input Parameters

alid	The identification number of the alarm to set or reset.
set	TRUE if the alarm should be set; otherwise, FALSE.

Output Parameter

status	0	Success
	1	The specified alarm record does not exist
	< 0	Standard AIM database status return code

Details

File: GEMDBASE.SQU

Related Routines

gm.alarm
gm.ed.alarm
gm.get.alarm
gm.raw.alarm

Calling Sequence

```
CALL gm.streamN ($raw, hd[ ], $data[ ], arrow)
```

Function

Process messages for a SECS-II message stream.

Usage Considerations

This description is for all the routines **gm.stream1( )**, **gm.stream2( )**, etc., where the number represents the stream number of the message that is being processed when this routine is called.

These programs exist for all the SECS-II message streams that are supported by the baseline AdeptGEM system (see **Table 5-1**). A new program with the appropriate name must be created to add support for a new SECS-II message stream. See **gm.user.strmN( )** on page 176 for information on the program parameters. (The defined streams are 1, 2, 5, 6, 7, 9, & 10.)

> **NOTE:** The routines **gm.streamN( )** do not use the **bypass** parameter.

Input Parameters

See **gm.user.strmN( )** on page 176.

Output Parameters

See **gm.user.strmN( )** on page 176.

Details

When the SECS-II task receives a message, the AdeptGEM software extracts the stream number from the message and looks for a program named with the format **gm.streamN**, where “**N**” represents the stream number. If the program is found, that program is CALLed and passed the message parameters. If the program is not found, an unknown-stream error message is returned (S9F3).

If you want to add support for a new SECS-II message, you should first look to see if the corresponding **gm.streamN** program already exists. If it does, you should modify or create the corresponding program **gm.user.strmN** to support the new message.

> **NOTE:** You should make sure that the routine **gm.user.strmN** returns with the **bypass** parameter set TRUE when your new message is processed.

If there is no existing program to process the message stream, you will need to create a new program named **gm.streamN**, where “**N**” represents the stream number, which contains the code necessary to process your new message. Thus, to add/modify functions for existing streams, use **gm.user.strmN**. To add/modify functions to streams that have not been defined, use **gm.streamN**.

File: GEMSTRM.SQU

Related Routines

gm.user.strmN

Calling Sequence

```
CALL gm.user.init ()
```

Function

Provide a convenient place for customizer initialization to be performed.

Details

This routine is called after all needed GEM global variables are initialized but before any background GEM tasks are started (if any). This routine is delivered as an empty shell. You can edit the routine to include any initialization data required for your custom installation.

File: GEMUSER.V2/SQU

Calling Sequence

```
CALL gm.user.mnu.ctl (type, condition1, condition2, condition3)
```

Function

Control user access to the AIM menu items for accessing the AdeptGEM databases.

Usage Considerations

This routine is called at numerous places in the baseline AdeptGEM code with **type** set to 0. Currently that has no effect, because there is no active code in the routine.

If a menu item has already been selected before this routine is called to dim that item, calling this routine will have no effect on the already-open menu page.

This routine could be called from the routine **cu.set.mode( )** to control menu access based on sequence execution. However, that might be done more effectively by modifying the definitions of the menu items, which is done in the routine **gm.mod.init( )**.

Input Parameter

type
Type of condition to check, as follows:
0 = Related to communication status
1 - 9 = Reserved for use by Adept code

condition1
Condition information, as follows:
For type = 0 (communication):
–1 = Communication is enabled
0 = Communication is disabled
[This parameter could be the global variable **gm.service.go**]

condition2
Condition information, as follows:
For type = 0 (communication):
–1 = Communicating
0 = Not communicating
[This parameter could be the global variable **gm.comm**]

condition3
Condition information, as follows:
For type = 0 (communication):
–1 = On-line
0 = Off-line
[This parameter could be the global variable **gm.online**]

Details

The AdeptGEM databases are accessed by the operator through items in the AIM Edit menu. Dimming (that is, disabling) those menu items may be desirable in order to maintain the integrity of database contents. For example, operator access could be prohibited if GEM communications are enabled, active, and/or on-line.

An example of how this routine operates is included in the distribution file GEMUSER.V2 as commented-out code.

File: GEMUSER.V2/SQU

Calling Sequence

```
CALL gm.user.packval (vid, $stream[], arrow, bypass, status)
```

Function

This routine determines the value of a custom status variable or data value.

Usage Considerations

The calling sequence in the distribution files is not correct (the calling sequence shown on the “CALL” line above is correct). If you are modifying this routine for custom processing and changing the bypass parameter to –1, you must also change the calling sequence so that it matches the “CALL” line above.

Input Parameters

vid The identification number of the variable for which information is desired.

$stream[ ] Array of strings representing the current packed contents of the SECS-II message that is being built.

arrow Position in the stream where the next item should be packed.

bypass Always set to FALSE.

Output Parameters

$stream[ ] Modified stream that contains an additional data item if the pack was successful; otherwise, it contains the unaltered stream.

arrow Position in the stream immediately following the data item that was packed (or, in case of error, the most recent successfully packed data item).

bypass Set to TRUE by this routine if the standard processing for the specified variable should be skipped.

status Status of the operation:
0 Success
<0 Standard AIM error code

Details

This routine is used for custom processing of GEM variables. Typically, this routine would be used to determine the value of a custom status variable or data value.

This routine is called before the standard processing is done for GEM variables that do not have a special Adept definition.

Using the variable ID, this routine can access the GEM variable database to obtain information about the variable that is to have its value packed.

This routine is expected to pack the variable value into a SECS-II message that is being built. Depending on the value to be packed, the appropriate routine **s2.pack.∗( )** can be used to do the actual packing of the value.

When adding new DVVALs, the user needs to define the variable in the GEM Variables database (as a String Function or Numeric Function) and add code to this routine. An example of the required code is shown below:

```
AUTO computed_value, $computed_string

status = 0                 ;Assume success

CASE vid OF

   VALUE 1001              ;Private real DVVAL for...

      computed_value = ...
      CALL s2.pack.1real(computed_value, ^44, $stream[], arrow)
      bypass = TRUE        ;Tell caller that the VID was processed.

   VALUE 1002:             ;Private string DVVAL for . . .

      $computed_string = ...
      CALL s2.pack.1ascii($computed_string, ^20, $stream[], arrow)
      bypass = TRUE        ;Tell caller that the VID was processed.

   VALUE 1003:             ;Obsolete DVVAL

      bypass = TRUE        ;Tell caller that the VID was processed.
      status = -401        ;Report undefined value.

END
```

File: GEMUSER.V2/SQU

Related Routines

gm.get.str
gm.get.val
gm.get.var
gm.pack.val
gm.var2val
s2.pack.1ascii
s2.pack.1bin
s2.pack.1real
s2.pack.list
s2.pack.null
s2.pack.reals
s2.pack.str

Calling Sequence

```
CALL gm.user.rc ($name, $argname[], $arg[], num, argstat[],
                                                    status)
```

Function

This routine can be used for customized processing of the GEM Remote Control additional capability.

Usage Considerations

This routine can be used to add or modify processing of remote commands received from the host. Unlike the routine **user.remote.cmd( )**, which is invoked for only a specific remote command, this routine is called for every remote command that is received.

This routine must *not* be deleted from memory even if it is not being used for custom processing.

Input Parameters

$name — Name of the “remote” command to be processed by this routine.

$argname[ ] — Argument names (elements 0 to **num**–1):
[0] = Name of argument 1
[1] = Name of argument 2

$arg[ ] — Argument values (elements 0 to **num**–1):
[0] = Value of argument 1
[1] = Value of argument 2

num — Number of elements in **$argname[ ]** and **arg[ ]**.

status — Always initially set to –1.

Output Parameters

argstat[ ] — Element N reports the status of the parameters **$argname[N]** and **$arg[N]**:

0	Both parameters are okay
1	One or both parameters does not exist
2	One or both parameters has an illegal value

status — Completion status of the request:

<0	Signal the calling routine that it should do its normal processing
0	Okay, the command has been performed
1	Command is not known
2	Cannot perform the command right now
3	At least one parameter is invalid (at least one **argstat[N]** <> 0)
4	Command will be performed with completion signaled later by an event
5	Rejected, already in desired condition

Details

This routine is called (by **gm.remote.cmd**) whenever an S2, F41 message is received from the host. The routine can be used to modify the command parameters in preparation for normal processing—in which case the **status** parameter should not be modified by this routine. Or, it can actually perform the requested command—in which case the **status** parameter should be set to zero or a positive value.

The input parameters for the routine are sent from the host using an S2F41 message in the following format:

```
L, 2
   1. “USER_CMD” ($name)
   2. L,n
      1. L,2
         1. “PARAM1_NAME” ($argname[0])
         2. “PARAM1_VALUE” ($arg[0])
      2. L,2
         1. “PARAM2_NAME” ($argname[1])
         2. “PARAM2_VALUE” ($arg[1])
      .
      .
      n. L,n
         1. “PARAMn_NAME” ($argname[num–1])
         2. “PARAMn_VALUE” ($arg[num–1])
```

> **NOTE:** Interpretation of the data within the program is the programmer’s responsibility.

If REMOTE/LOCAL mode restrictions are desired within the routine, use the global variable **gm.remote** to determine the mode that the system is in.

> **WARNING:** The variable **gm.remote** should only be read and *never* modified in this routine.

File: GEMUSER.V2/SQU

Related Routines

gm.remote.cmd
user.remote.cmd

Calling Sequence

```
CALL gm.user.reply (hd[], $data[], instatus, bypass)
```

Function

Handle replies for host messages that were frozen (queued) by the equipment and later sent to the host.

Input Parameters

hd[ ] The header information for the primary message that was sent out (thus the function number of the reply will be one more than **hd[uc.hd.function]**).

$data[ ] The actual body of the message received from the host (if instatus is 0).

instatus The status received from **gm.usersend( )**. This is normally 0 for success, but in case of an error, it will have a special or V$^+$ error code. This subroutine may be interested if an attempt to send a primary message has failed, and with this parameter, it can take appropriate action.

bypass Always set to FALSE.

Output Parameters

hd[ ] May be modified by this routine.

$data[ ] May be modified by this routine.

instatus May be modified by this routine.

bypass Boolean set to TRUE by this routine to completely bypass the normal processing of this message.

Details

Since the calling program doesn't care about the reply (if it did, it would have waited for the reply rather than just frozen it and moved on), we handle the reply here. In most cases, this just involves making sure the host sent back an acknowledge code as expected — and we make note what happened in the debug log. In some cases, further action is needed, such as S1F2, which implies a successful on-line (to prevent stalling, S1F1 is queued rather than sent-and- waited for).

You can modify this routine to change the responses to already implemented replies. New responses require that the response code be written and the response sent with **gm.freeze.msg( )**.

File: GEMUSER.V2/SQU

Related Routines

gm.freeze.msg
gm.usersend

Calling Sequence

```
CALL gm.user.strmN ($raw, hd[], $data[], arrow, bypass)
```

Function

Process host messages that are not supported by the baseline AdeptGEM system, or modify the standard processing of messages that are supported.

Usage Considerations

This description is for all the routines **gm.user.strm1**, **gm.user.strm2**, etc., where the number represents the stream number of the message that is being processed when this routine is called.

These routines can be used to add or modify processing of functions in message streams that are already supported by the baseline AdeptGEM system. See **Table 5-1** for the steams and functions that are supported.

> **CAUTION:** These routines *must not* be deleted from memory even if they are not being used for custom processing.

Input Parameter

$raw The original header received in the SECS-II primary message from the host. Generally this will be used only if a stream-9 message is generated in response to the received message (see below).

hd[ ] Header data that has been extracted from the received message. The following array elements may be significant to this routine:

- hd[uc.hd.wait] Boolean that is TRUE if the host expects a reply message (i.e., if the W bit was set in the received message)
- hd[uc.hd.stream] Stream number for the received message (always the same as the number in the name of this routine)
- hd[uc.hd.function] Function number for the _reply_ message (i.e., one larger than the function number that was in the received message)

$data[ ] The body of the received message (i.e., without the header). The routines **s2.unpack.*** can be used to extract formatted information from this “raw” data.

arrow Always set to 1 to facilitate processing of the contents of the $data[] array. This variable can be used with the **s2.unpack.*** routine to keep track of the progress of extracting information from the **$data[ ]** array.

bypass Boolean that is always initially set FALSE.

Output Parameter

hd[ ] Header data for the reply message to send, if any. Normally this array is not modified by this routine. The element **hd[uc.hd.wait]** must have a nonzero value if a reply message is to be sent to the host.

$data[ ] Normally this array contains the body of the reply message to send, if any. The routines **s2.pack.*** can be used to create the contents of this array. (Note that all the input data must be extracted from this array before the reply message is composed. Alternatively, the reply message can be composed in a separate, local array variable, and then copied to this array at the end of the routine.)

In some situations it might be desirable to have this routine simply modify the received contents of this array in preparation for normal processing by the baseline AdeptGEM system. In that case, the **arrow** parameter should return the value 1, and the **bypass** parameter should be set to FALSE.

arrow The character position in the **$data[ ]** array that follows the last character of the reply message. That is, the contents of the **$data[ ]** array from this point on will be ignored. (Note that if the input message is processed while the reply message is being composed, you must maintain the reply pointer as a separate, local variable until the whole input message has been processed. Then you must assign the value of that variable to this parameter.)

bypass Boolean set TRUE by this routine to signal the normal message processing that the message has already been completely processed.

Details

One of these routines is called for every received host primary message in the corresponding stream N group that is supported by the baseline AdeptGEM system. Thus, you can add code to these routines to support additional functions, or to modify the standard processing of functions that are already supported by the baseline AdeptGEM system.

> **NOTE:** If your custom code completely processes the message (i.e., it performs all the appropriate operations and composes the reply message, if any), the **bypass** output parameter must be set TRUE to signal the normal message processing that the message has already been completely processed.

> **CAUTION:** Changing any responses defined by the baseline AdeptGEM system may result in a GEM system that is not SEMI compliant. The user is responsible for ensuring (and documenting) SEMI compliance of any added or altered messages.

If you want to add support for functions in a message stream that is not supported by the baseline AdeptGEM system, you need to create a routine named **gm.streamN** (see page 168), where N represents the number of the new stream (for example, **gm.stream4**). In that case, you do *not* need a corresponding routine named **gm.streamN**.

If this routine generates a stream-9 error message in response to the received message, the following code would normally be used to prepare that message (in this case for an S9F5 message):

```
        $data[0] = $raw              ;Copy received header
        arrow = LEN($raw)+1          ;Indicate length of the data
        CALL gm.new.head(9, 5, hd[]) ;Compose new header for S9F5 message
        hd[uc.hd.wait] = TRUE        ;Signal there's a reply to send
```

File: GEMUSER.V2/SQU

Related Routine

gm.streamN

Calling Sequence

```
CALL gm.usersend (reply, hd[], length, $data[], timer, status)
```

Function

Send a primary message to the host, and, if a reply is desired, wait for a reply (or a timeout error).

Many associated low-level details are handled automatically by this subroutine, allowing the calling program to be as simple as possible. This includes:

- If communications are not enabled, this subroutine will return with an error.
- If communications are enabled, but the host is not communicating, this subroutine will wait for up to 20 seconds to try to send the message (and if it still can't send it, it will return with an error).
- If the primary message requires an inquire/grant message first (for example, a multi-block S7F3 message needs S7F1 to be sent and acknowledged first), then the inquire/grant is automatically processed. The main primary message is sent only if the host grants acceptance (otherwise, this subroutine returns with an appropriate AIM error code).

Input Parameters

reply — TRUE if a reply is desired; otherwise, FALSE.

hd[ ] — An array representing the header for the message, with the “uc.hd.stream”, “uc.hd.function” and “uc.hd.sys.#” elements filled. The routine **gm.new.head( )** is the most appropriate way to create this header.

length — The total length, in bytes, of the message.

$data[ ] — The stream of bytes, fully PACK'ed, representing the message. The “**s2.pack.***” routines can be used to create this stream of bytes.

Output Parameters

$data[ ] — The stream of bytes, fully PACK'ed, representing the host's reply to the primary message that was sent out. (This reply overwrites the message that was sent out.) A reply exists only if **status** is 0.

timer — Normally 0, but non-zero if **status** is equal to **uc.ietmo**. Then, this number indicates which timer caused a timeout error (3 for T3, 4 for T4, etc.)

status — Zero if success, or a standard AIM error code if an error occurred.

Details

Messages are sent regardless of the current control state (if this is significant to the calling program, the calling program should, for example, check the global variable **gm.online**). This subroutine will get confused if the equipment receives a message from the host *other* than the expected reply for the message sent out. Multiple open transactions are not supported.

This subroutine may take a long time to execute if there is a problem sending the message or if the reply takes a long time. If the calling program does not care too much about the reply, and if it is more important that the calling program continue running without a big delay, **gm.freeze.msg( )** can be used instead. This will queue the message, which will be sent by the service task at the next opportunity.

Any replies received from the host will be handled by the following routines:

gm.reply( ) This is an internal routine that processes replies that are received from the host in response to (primary) messages sent by the equipment.

gm.user.reply( ) This is called by **gm.reply( )** before doing any standard processing of the response message. Similar to **gm.user.strmN( )**, **gm.user.reply( )** can be used to modify/ replace the normal processing done in **gm.reply( )**.

File: GEMENGC.SQU

Related Routines

gm.com.test
gm.estcom
gm.freeze.msg
gm.new.head
gm.send.hostmsg
gm.user.reply
s2.pack.1ascii
s2.pack.1bin
s2.pack.1real
s2.pack.list
s2.pack.null
s2.pack.reals
s2.pack.str

Calling Sequence

```
CALL gm.var2val (vid, oldval, vtype, gtype, newval, status)
```

Function

Convert the value returned from the Value field of a record in the GEM Variables database to the actual value of the variable.

Input Parameters

vid	ID for the variable being considered (this parameter is used only for the V^+ variable **vtype**).
oldval	The value that is to be (potentially) converted, based on the type specified.
vtype	The type of the variable the specified value is associated with.
gtype	GEM Variables class for the variable.

Output Parameters

newval	The converted, actual value of the variable.
status	0 upon success, or 1 if a conversion could not be made because of an inappropriate input value.

Details

When the Value field is read from the GEM Variables database (e.g., with **gm.get.var( )**), it does not necessarily represent the actual value of the variable. It may, for example, represent an **ai.ctl[ ]** index, and the array element specified by the index would hold the actual value. This subroutine will convert a “raw” value into an actual value, based on the variable type given.

An example conversion failure would be a negative input value and a variable type of **gv.type.ctlv**(6) — this would be interpreted as an **ai.ctl[ ]** with a negative index, which is illegal.

A conversion failure is returned if the **gtype** parameter has the value **gv.cls.datval**, indicating that the variable represents a Data Value.

File: GEMENG.SQU

Related Routines

gm.get.val
gm.get.var
gm.set.var
gm.user.packval

Calling Sequence

```
CALL line_check (args[], error)
```

Function

Statement execution routine for the LINE_CHECK statement. It is used to check the status of the equipment-to-host connection.

> **NOTE:** The statement name is hard-coded in the routine **gm.scr.spawn( )** (in the file GEMAIM.V2). That routine shows only numeric variables in the scrolling pick list.

Usage Considerations

□ **Runtime** □ **Control** □ **Robot** □ **Vision**

Statement Syntax

```
LINE_CHECK --gemvariable-- = COMM_STATUS
```

Input Parameters

args[ ] Real array containing the arguments for this statement (record numbers or constants). The individual elements are described below:

`--gemvariable--` `args[1]`
Record number from the GEM Variables database.

Output Parameters

error Communication status error code stored in the specified GEM variable (if it is a valid destination). The value will be 0 if the host correctly acknowledged the message sent to it. Otherwise, a standard AIM error response code is returned. See the standard AIM operator error response code values for details.

The most common error would be –531 (*Communication time-out*), indicating that the host is not responding. If communications are not enabled, the error code –508 (*Device not ready*) is used.

Details

This statement forcibly sends an S1F13 message to the host to verify its presence and ensure that it is responding to messages properly. If not, an error will be returned in the output error argument. The S!F13 message is sent even if the equipment is not On-Line. (This does not conflict with the SEMI standards, since S1F13 is used for establishing communications.)

If there is a communication problem, the calling program could be seriously delayed. For example, if the communications state is "Attempt On-Line" and the service and communication tasks are already trying to talk with the host, this routine will wait 20 seconds before giving up. Or, if the state is "Communicating" but the host never sends a reply message, 45 seconds could elapse before this subroutine returns with an error.

The output GEM variable must have the type "numeric variable".

File: GEMSTMT.SQU

Statement DB: STATGEM.DB

Related Routines

Calling Sequence

```
CALL run_check (args[], error)
```

Function

Statement execution routine for the RUN_CHECK statement. It is used to check whether or not it is okay for the current sequence to command robot motion.

> **NOTE:** The statement name is hard-coded in the routine **gm.scr.spawn( )** (in the file GEMAIM.V2). That routine shows only numeric variables in the scrolling pick list.

Usage Considerations

□ **Runtime** ☑ **Control** □ **Robot** □ **Vision**

Statement Syntax

```
RUN_CHECK --gemvariable-- = MOTION_OK
```

Input Parameters

args[ ] Real array containing the arguments for this statement (record numbers or constants). The individual elements are described below:

```
--gemvariable--                                    args[1]
```
Record number from the GEM Variables database.

Output Parameters

error Real variable that receives a value indicating whether or not the operation was successful and what action should be taken by the calling routine. See the standard AIM operator error response code values for details.

A Boolean TRUE/FALSE is stored in the specified GEM variable (if it is a valid destination) indicating whether or not conditions permit the sequence to perform robot motion. See the Data Structure described in the Details section for more information.

Details

This routine is used in a sequence to verify that the current state of the equipment is compatible with host control of robot motion. It is considered okay in any of the following situations:

1. The sequence was initiated by the host and the control state is Remote. (The communication state must be On-Line for the host request to be received.)
2. The sequence was initiated by the user and one of the following is true:
    a. The communication state is “Not Communicating”.
    b. The control state is “On-Line/Local”.
    c. The control state is “Off-Line/Equipment Off-Line”.

The output GEM variable must have the type “numeric variable”.

Data Structure:

gm.comm	Boolean TRUE if communicating
gm.online	Boolean TRUE if communication is On-Line (*)
gm.remote	Boolean TRUE if communication is Remote (*)
gm.stask	Number of task used for GEM “service”
gv.db[ ]	Number of the GEM Variables database
gm.gv.lock	Lock variable for access to the GEM Variables database

(*) The value of this variable should not be considered if not communicating (i.e., if “gm.comm” is FALSE). In that case, this variable indicates the state that will result the next time communication is established.

File: GEMSTMT.SQU

Statement DB: STATGEM.DB

Related Routines

Calling Sequence

```
CALL s2.bytes2float ($string, value)
```

Function

Convert a 4-byte IEEE floating- point number, represented by a sequence of bytes in a V^{+} string, to a V^{+} single-precision real value.

Input Parameter

$string	The string containing the sequence of four bytes that represent an IEEE-format floating-point number.

Input Parameter

value	The single-precision real value obtained.

Details

No error checking is performed to ensure the input string is 4 bytes in long.

File: GEMS2.SQU

Related Routine

s2.bytes2int

Calling Sequence

```
CALL s2.bytes2int ($bytes, signed, value)
```

Function

Convert a raw sequence of bytes, represented as a V$^+$ string, into a V$^+$ integer value. Any number of bytes can be used. If the string contains more than one character, BIG ENDIAN is used (i.e., the most significant byte is processed first).

Input Parameters

$bytes	The byte(s) to be converted
signed	TRUE if the byte sequence is to be considered a signed integer instead of an unsigned integer

Output Parameter

value	The resulting integer value

Details

This routine is similar to the V$^+$ function INTB( ). However, this routine will process a string of any length and will convert unsigned as well as signed integers.

File: GEMS2.SQU

Related Routine

s2.bytes2float

Calling Sequence

```
CALL s2.pack.1ascii ($ascii, $stream[], arrow)
```

Function

Pack a single ASCII data item into a SECS-II data stream.

Input Parameters

$ascii	The string to be packed.
$stream[ ]	The stream of bytes to write to receive the data item.
arrow	The byte position in the stream to start writing data to. The first byte in the stream is number 1.

Output Parameters

$stream[ ]	The modified stream of bytes.
arrow	The byte position in the stream just following the last byte packed.

Details

The **arrow** input parameter represents the character position that data should be written to in the stream, and it is automatically incremented by the number of bytes packed. The data item packed is always of SEMI data type ^20.

The maximum string size that can be packed is 128 characters. Larger strings can be written if the routine **s2.pack.str( )** is used.

File: GEMS2.SQU

Related Routines

s2.pack.str
s2.unpack.str

Calling Sequence

```
CALL s2.pack.1bin ($bits, $stream[], arrow)
```

Function

Pack a string representation of a binary value as a binary data item into a SECS-II data stream.

Input Parameters

$bits	The string of ASCII "0" and "1" characters representing the binary value, starting with the *least* significant bit.
$stream[ ]	The stream of bytes to receive the data item.
arrow	The byte position in the stream to start writing data to. The first byte in the stream is number 1.

Output Parameters

$stream[ ]	The modified stream of bytes.
arrow	The byte position in the stream just following the last byte packed.

Details

The **arrow** input parameter represents the character position that data should be written to in the stream, and it is automatically incremented by the number of bytes packed. The data item packed is always of SEMI data type ^10.

The **$bits** input parameter contains a string of ASCII "0" and "1" characters representing the binary data to be packed. The first character of the string defines the setting of bit #1 (the least-significant bit), the second character defines bit #2, etc. For example, the string "11010000" will be packed as one byte containing the binary value ^B1011.

The number of bytes in the binary data item is the length of the string divided by 8. (If the string ends with an incomplete set of "bits", the missing bits are assumed to be zero.)

Unpredictable results will occur if **$bits** contains characters other than "0" or "1". If **$bits** is empty, then a zero-length binary data item is packed.

File: GEMS2.SQU

Related Routines

s2.pack.1ascii
s2.pack.str

Calling Sequence

```
CALL s2.pack.1real (value, type, $stream[], arrow)
```

Function

Pack a single real value into a SECS-II data stream represented by **$stream[ ]** and **arrow**.

Input Parameters

value	The real value to be packed.
type	The SECS-II data type to use to represent the value. If bit #7 (mask ^100) is set, then a list of one item is packed instead of a single non-list item (see Details).
$stream[ ]	The stream of bytes to receive the data item.
arrow	The byte position in the stream to start writing data to. The first byte in the stream is number 1.

Output Parameters

$stream[ ]	The modified stream of bytes.
arrow	The byte position in the stream just following the last byte packed.

Details

The 'arrow' input parameter represents the character position that data should be written to in the stream, and it is automatically incremented by the number of bytes packed.

Data formatting is automatic for these supported SEMI data types: ^10, ^11, ^20, ^31, ^32, ^34, ^44, ^51, ^52 and ^54. (Data types ^21, ^30, ^40 and ^50 are not supported. No error results if one of the unsupported data types is specified, but data type ^44 is used by default.)

When data type ^10 is used, the value is represented by eight bits. When data type ^11 is used, any nonzero value is considered TRUE.

When a numeric value is packed (i.e., for all the data types other than ^11 and ^20), if the value to be packed exceeds the possible range of values for the data type, the routine uses a different data type that can “accommodate” the value.

The data value can be packed either as a simple numeric item, or as a list with one item. There are two ways to request a list:

1. The data is packed as a list if the 'type' parameter has the value 0, explicitly requesting a list. In this case, the list item is formatted with the default data type, ^44.
2. The data is packed as a list if the 'type' parameter has bit #7 (i.e., mask value ^100) set. In this case, the list item is formatted with the data type specified by the low-order six bits of the 'type' parameter. (That is, add ^100 to the desired item format type to get a list of items with that type.)

File: GEMS2.SQU

Related Routines

s2.pack.reals
s2.unpack.1real
s2.unpack.real

Calling Sequence

```
CALL s2.pack.list (items, $stream[], arrow)
```

Function

Pack a list header into a SECS-II data stream.

Input Parameters

items	The number of items in the (forthcoming) list. The maximum allowed value is 65,535; the minimum value is 0. See the Details section for more information.
$stream[ ]	The stream of bytes to receive the list header.
arrow	The byte position in the stream to start writing data to. The first byte in the stream is number 1.

Output Parameters

$stream[ ]	The modified stream of bytes.
arrow	The byte position in the stream just following the last byte packed.

Details

This routine can be used to pack a list header, in anticipation of packing the individual list items. The 'items' parameter represents the number of items that will be packed in the list; it is up to the calling program to pack those items after this routine is called. The 'arrow' parameter represents the character position at which the data should be packed into the stream, and it is automatically incremented by the number of bytes that are packed.

In some situations the number of items in a list is not known before the list items are packed. In order to simplify the processing of such cases, the 'items' parameter can be given a non-integer value[1] to force the use of a 2-byte list-length value in the list header. (That reserves space in the list header for the maximum number of list items.) Then, after the list items are packed, the actual length of the list can be “plugged in” the previously packed list header. The specific steps that the calling program would use for this process are as follows:

1. Save the value of 'arrow' (SA).
2. Call this routine with a “signal” value for the number of list items. This packs a list header with a 2-byte list-length value (that contains zero).

   ```
   CALL s2.pack.list(0.5, $stream[], arrow)
   ```

   If an expected number of list items is known, that value (plus 0.5) can be used in this CALL. Then, if it that number turns out to be correct, step 5 below can be skipped.
3. Pack all the individual list items.
4. Determine the actual number of list items that were packed (N).

[1] The fractional part of the non-integer value must be greater than 0.25.

5. Call this routine again, supplying the actual number of list items and the saved value of 'arrow'. This overwrites the previous list header with one containing the correct list length.

   ```
   CALL s2.pack.list(N+0.5, $stream[], SA)
   ```

   Note that 0.5 must be added to the actual list length to make sure that the replacement header also uses a 2-byte list-length value.

File: GEMS2.SQU

Calling Sequence

```
CALL s2.pack.null (type, $stream[], arrow)
```

Function

Pack a zero-length item of any specified type into a SECS-II data stream represented by **$stream[ ]** and **arrow**.

Input Parameters

type	The SECS-II data type of the zero-length item to pack.
$stream[ ]	The stream of bytes to receive the data item.
arrow	The byte position in the stream to start writing data to. The first byte in the stream is number 1.

Output Parameters

$stream[ ]	The modified stream of bytes.
arrow	The byte position in the stream just following the last byte packed.

Details

The **arrow** input parameter represents the character position that data should be written to in the stream, and it is automatically incremented by the number of bytes packed.

File: GEMS2.SQU

Calling Sequence

```
CALL s2.pack.reals (values[], start, num, type, $stream[], arrow)
```

Function

Pack an array of real values into a SECS-II data stream.

Input Parameters

values[ ]	The array of real values to be packed.
start	The number of the first element of **values[ ]** to use, typically 0 or 1.
num	The total number of elements of **values[ ]** to pack.
type	The SECS-II data type to use to represent the values. If bit #7 (mask ^100) is set, a list of values is packed instead of a single array item (see Details).
$stream[ ]	The stream of bytes to receive the data items.
arrow	The byte position in the stream to start writing data to. The first byte in the stream is number 1.

Output Parameters

$stream[ ]	The modified stream of bytes.
arrow	The byte position in the stream just following the last byte packed.

Details

The 'arrow' input parameter represents the character position that data should be written to in the stream, and it is automatically incremented by the number of bytes packed.

The elements of the 'values[]' array can be packed either as a single array data item with 'num' values, or as a list with 'num' items. There are two ways to request a list:

1. The data is packed as a list if the 'type' parameter has the value 0, explicitly requesting a list. In this case, the list items are formatted with the default data type, ^44.
2. The data is packed as a list if the 'type' parameter has bit #7 (i.e., mask value ^100) set. In this case, the list items are formatted with the data type specified by the low-order six bits of the 'type' parameter. (That is, add ^100 to the desired item format type to get a list of items with that type.)

A single data item with no value, or a zero-length list, can be packed by setting 'num' to zero. All the array elements are packed using the same data type. If the SECS-II message requires that different data types be used for different array elements, the elements will have to be packed in the stream individually, possibly with multiple calls to this routine.

Data formatting is automatic for these supported SEMI data types: ^10, ^11, ^20, ^31, ^32, ^34, ^44, ^51, ^52 and ^54. (Data types ^21, ^30, ^40 and ^50 are not supported. No error results if one of the unsupported data types is specified, but data type ^44 is used by default.)

When data type ^10 is used, each value is represented by eight bits. When data type ^11 is used, any nonzero value is considered TRUE.

File: GEMS2.SQU

Related Routines

s2.pack.1real
s2.unpack.1real
s2.unpack.real

Calling Sequence

```
CALL s2.pack.str ($str[], start, len, type, $stream[], arrow)
```

Function

Pack an array of strings into a SECS-II data stream represented by $stream[ ] and arrow.

Input Parameters

$str[ ] The array of strings to be written.

start The number of the first element of **$str[ ]** to use, typically 0 or 1.

len The total number of elements of **$str[ ]** to pack.

type The SECS-II data type to convert to. If bit #7 is set, then a list of values instead of a single array item is written (see abstract).

$stream[ ] The stream of bytes to write to.

arrow The byte in the stream to start writing data to. The first byte in the stream would be 1.

Output Parameter

$stream[ ] The modified stream of bytes.

arrow The byte in the stream just following the last byte written.

Details

arrow represents the character that data should be written to in the stream, and it is automatically incremented after the data is packed by the number of bytes written. Data conversion is automatic. SEMI supported types are: ^10, ^11, ^20, ^31, ^32, ^44, ^51, ^52.

Data type not supported are: ^21, ^30, ^34, ^40, ^50, ^54.

If bit #7 of type is set (i.e. 64 is added to the type), then the data will be packed as a list with one array element in one item instead of the normal algorithm in which an entire array is packed into one item.

All elements written are converted to the same type. If SECS-II requires that different types be used for different array elements, then the elements will have to be written

to the stream individually, possibly with multiple calls to **s2.pack.str( )**. If an unsupported type is used, then no error is given, but type ^51 is used by default.

Conversion information:

^10 A single string of 0s and 1s is converted to raw binary data. The first byte of the string is bit #1, the second is bit #2, etc. Any additional strings are always ignored unless a SECS-II list is requested, in which case each element of the array becomes one data item.

^11 String should be “TRUE” or “FALSE”

^20 Normally, a single string is used directly, up to128 characters. If a longer string is desired, subsequent array elements can have a continuation of the string if each continuation string begins with ASCII value 0. Any non-continuation strings in the array are ignored if not the first, unless a SECS-II list is requested, in which case the entire array is packed, each into a separate data item, with the continuation strings still being taken into account.

Example (assume $ is the ASCII 0 character):

```
$str[0] = "Hello "

$str[1] = "$World"

$str[2] = "AdeptGEM"
```

In non-list mode, a single data item is packed of type ASCII with value “Hello World”. In list mode, a 2-element list is packed with each element being of type ASCII. The values are “Hello World” and “AdeptGEM” respectively. Notice that in the first (non-list) case, $str[2] is ignored.

Numbers The function VAL( ) is used on each array element. Only one number is read per element. Zero-length lists or items can be created by setting **len** to 0. If packing ASCII strings (type ^20), then zero- length strings can also be used.

File: GEMS2.SQU

Related Routines

s2.pack.1ascii
s2.unpack.str

Calling Sequence

```
CALL s2.scan.header ($stream[], arrow, format, length)
```

Function

Scan the header of a SECS-II data item.

Input Parameters

$stream [ ]	The input stream of bytes.
arrow	The character position in the stream to read from next (the first byte in the stream is number 1).

Output Parameters

arrow	The character position in the stream of the body of the data item examined.
format	The SECS-II format code extracted from the header.
length	The length (in bytes) of the item body, except in the case of a list, when the length represents the number of items in the list. In case of an error in the header, length is set to –1.

Details

From an input stream of bytes, represented by **$stream[ ]** and **arrow**, the SECS-II format code and the length of the data item body are obtained. The stream pointer is incremented the correct amount to point at the start of the body of the data item.

The format code is extracted but not checked for validity. It is up to the calling program to interpret the results returned by this function.

If **arrow** is passed by value, it will not be changed by this routine. That way, this routine can be used to determine which extraction routine(s) will be needed to gather the data from the stream.

File: GEMS2.SQU

Calling Sequence

```
CALL s2.stream.trim (position, $stream[])
```

Function

Remove trailing, unwanted characters from the end of a SECS-II data stream.

Input Parameters

position	The character position at which to start removing characters.
$stream[ ]	The stream to remove extra characters from.

Output Parameter

$stream[ ]	The modified (“trimmed”) stream.

Details

The **pos** parameter is the position of the first unwanted character in the stream, which is one more than the desired length of the stream. In the last 128-byte element of the stream array that is retained, all characters starting at **pos** are removed, and the next element of the stream is made empty.

Only the unwanted characters in the last element and the next element after that are erased. If any characters exist in later elements, they are not removed. However, enough characters are removed so that most functions should be able to deal with the stream correctly.

File: GEMS2.SQU

Calling Sequence

```
CALL s2.unpack.1real ($stream[], arrow, value, status)
```

Function

Extract a numeric value from a SECS-II data stream.

Usage Considerations

This routine should normally only be used when you know in advance that only one item should appear. See **s2.scan.header( )** for information on how to look ahead at a data item.

Input Parameters

$stream[ ]   The input stream of bytes.

arrow   The character position in the stream to read from next.

Output Parameters

arrow   The location of the next item in the stream, if any.

value   The real value obtained from the data item.

status   Status indicator:
0 Real-value extraction was successful
1 One or more items couldn't be converted
2 There is an error in the stream
3 The stream was OK, but no value was extracted (possible for a zero-length list)

Details

This subroutine works exactly like **s2.unpack.real( )**, except that it only obtains one number rather than an array of numbers. All the list-collapsing functionality of **s2.unpack.real( )** is still available with this routine. However, if more than one value is extracted from the next item in the stream, only the first value is returned.

File: GEMS2.SQU

Related Routines

s2.pack.1real
s2.pack.reals
s2.unpack.real

Calling Sequence

```
CALL s2.unpack.real ($stream[], arrow, pos, val[], status)
```

Function

Extract a real value or array of real values from a SECS-II data item.

Input Parameters

$stream[ ] The input stream of bytes.

arrow The character position in the stream to read from next.

pos The element number of the output array to receive the first real value. Subsequent values are stored in higher-numbered elements.

Output Parameters

arrow The location of the next item in the stream, if any.

pos One greater than the highest array element of **val[ ]** used. This value can help indicate how many real values were actually extracted.

val[ ] The real value(s) obtained from the data item.

status Status indicator:
0 Real-value extraction was successful
1 One or more items couldn’t be converted
2 There is an error in the stream

Details

The data item examined is at the byte position indicated by **arrow** in the stream of bytes **$stream[ ]**. After this function is called, **arrow** is incremented to the next item (unless there is an error in the stream). All conversions are performed automatically: this function will examine the item and convert to one or more real values from whatever SECS-II data type was found (unsigned 4-byte integer, 4-byte floating point, ASCII, etc.). The first real value is stored in **val[pos]**, and **pos** is incremented for subsequent values, if any.

This function has the ability to collapse a complicated, nested list structure into a linear array. This can be a tremendous advantage if the structure is in a predefined, standard format. On the other hand, the structure is lost if it is not already known. In this latter case, the calling program would need to scan at least part of the stream manually.

When an item of type ^10 is encountered, 8 bits from the data stream are stored per array element.

File: GEMS2.SQU

Related Routines

s2.pack.1real
s2.pack.reals
s2.unpack.1real

Calling Sequence

```
CALL s2.unpack.str ($stream[], arrow, pos, $str[], status)
```

Function

Extract a string or array of strings from a SECS-II data item.

Input Parameters

$stream[ ]	The input stream of bytes.
arrow	The character position in the stream to read from next.
pos	The element number of the output array to receive the first string. Subsequent strings are stored in higher-numbered elements.

Output Parameters

arrow	The location of the next item in the stream, if any.
pos	One greater than the highest array element of **$str[ ]** used. This value can help indicate how many strings were actually extracted.
$str[ ]	The string(s) obtained from the data item.
status	Status indicator: 0 String extraction was successful 1 One or more items couldn't be converted 2 There is an error in the stream

Details

The data item examined is at the byte position indicated by **arrow** in the stream of bytes **$stream[ ]**. After this function is called, **arrow** is incremented to the next item (unless there is an error in the stream). All conversions are performed automatically: this function will examine the item and convert to one or more strings from whatever SECS-II data type was used (unsigned 4-byte integer, 4-byte floating point, ASCII, etc.). The first string is stored in **$str[pos]**, and **pos** is incremented for subsequent values, if any.

This function has the ability to collapse a complicated, nested list structure into a linear array. This can be a tremendous advantage if the structure is in a predefined, standard format. On the other hand, the structure is lost if it is not already known. In this latter case, the calling program would need to scan at least part of the stream manually.

Conversion information:

^10	String of ASCII 0s and 1s in **reverse** order. The first character represents bit #1, the second character represents bit #2, etc.
^11	TRUE or FALSE
^20	If a string is longer than 128 bytes, additional array elements are used with an ASCII 0 character at the beginning of each "continuation" string.

File: GEMS2.SQU

Related Routines

s2.pack.1ascii
s2.pack.str

Calling Sequence

```
CALL send_host_msg (args[], error)
```

Function

Statement execution routine for the SEND_HOST_MSG statement. It is used to send a message to the host.

> **NOTE:** The statement name is hard-coded in the routine **gm.scr.spawn( )** (in the file GEMAIM.V2). That routine shows only string variables in the scrolling pick list.

Usage Considerations

□ **Runtime** □ **Control** □ **Robot** □ **Vision**

Statement Syntax

```
SEND_HOST_MSG --gemvariable--
```

Input Parameters

args[ ] Real array containing the arguments for this statement (record numbers or constants). The individual elements are described below:

`--gemvariable--` `args[1]`
Record number from the GEM Variables database.

Output Parameters

error Real variable that receives a value indicating whether or not the operation was successful and what action should be taken by the calling routine. See the standard AIM operator error response code values for details.

Details

This statement sends a text message to the host. For the message to actually be sent, the state must be Communicating and On-Line. However, no action is taken if there is an error.

When this routine is executing, it temporarily locks access to the GEM Variables database.

The output GEM variable must have the type “string variable”.

File: GEMSTMT.SQU

Statement DB: STATGEM.DB

Related Routines

Calling Sequence

```
CALL set_state (args[], error)
```

Function

Statement execution routine for the SET_STATE statement. It is used to set the value of a State variable to a State ID.

> **NOTE:** The statement name is hard-coded in the routine **gm.scr.spawn( )** (in the file GEMAIM.V2). That routine shows only states in the scrolling pick list.

Usage Considerations

□ **Runtime** □ **Control** □ **Robot** □ **Vision**

Statement Syntax

```
SET_STATE --gemvariable--
```

Input Parameters

args[ ] Real array containing the arguments for this statement (record numbers or constants). The individual elements are described below:

`--gemvariable--` `args[1]`
Record number from the GEM Variables database.

Output Parameters

error Real variable that receives a value indicating whether or not the operation was successful and what action should be taken by the calling routine. See the standard AIM operator error response code values for details.

Details

This statement sends a text message to the host. For the message to actually be sent, the state must be Communicating and On-Line. However, no action is taken if there is an error.

When this routine is executing, it temporarily locks access to the variables and string databases.

The output GEM variable must have the type “string variable”.

File: GEMSTMT.SQU

Statement DB: STATGEM.DB

Calling Sequence

```
CALL user.remote.cmd ($name, $argname[], num, argstat[], status)
```

Function

This is a user-written routine that is executed in response to a remote command received from the host.

You can give this routine any valid V^+ program name. The name you choose for this routine must match the name of a remote command that the host will send in an S2,F41 message.

Usage Considerations

The parameter list for this routine must match the number and types of parameters shown above, although you can use any variable names you wish.

This routine has the same parameter list as the routine **gm.user.rc** (which is provided in the file GEMUSER.V2). Thus, you can use a copy of that routine as a starting point for your user-written routine.

> **NOTE:** The routine **gm.user.rc** must not be deleted from system memory even if it is not modified.

You can create as many different remote-command routines of this type as needed.

Details

$name — Name of the “remote” command to be processed by this routine. (The value of this parameter will always be the same as the actual name of this routine.)

$argname[ ] — Argument names (elements 0 to num–1):
[0] = Name of argument 1
[1] = Name of argument 2

$arg[ ] — Argument values (elements 0 to num–1):
[0] = Value of argument 1
[1] = Value of argument 2

num — Number of elements in **$argname[ ]** and **$arg[ ]**

Output Parameter

argstat[ ] — Element N reports the status of input parameters **$argname[N]** and **$arg[N]**, as follows:
0 Both parameters are okay
1 One or both parameters does not exist
2 One or both parameters has an illegal value

status — Completion status of the request, as follows:
0 Okay
1 Command is not known
2 Cannot perform command right now
3 At least one parameter is invalid (i.e., at least one **argstat[ ]** <> 0)

Details

This routine is called (by **gm.remote.cmd**) when a remote command is received from the host (in an S2,F41 message) and the following conditions are satisfied:

1. The command is not processed by the user-modified routine **gm.user.rc**.
2. The command name does not match one of the commands that are predefined in the baseline AdeptGEM system.
3. The command name does match the name of this routine.

The user-modified routine **gm.user.rc** could also call this routine.

This routine should perform whatever steps are appropriate to process the requested remote command. Then the **status** parameter should be set to one of the values listed above, or any other value the host accepts as the HCACK acknowledge code in an S2,F42 message.

File: This is a user-written routine. It is not provided with the AdeptGEM module.

Related Routines

gm.remote.cmd
gm.user.rc

Calling Sequence

```
CALL user.state.spawn (vid, state_id)
```

Function

This is a user-written routine that will be executed in response to a state transition.

You can give this spawn routine any valid V$^+$ program name. The name you choose must match the name specified in the Spawn Routine field in a State record in the GEM Items database.

Usage Considerations

The parameter list for your routine must match the number and type of parameters shown above, although you can use any variable names you wish.

You can create as many different spawn routines of this type as needed.

Input Parameters

vid	Associated VID (record from the GEM Variables database)
state_id	ID of a State record in the GEM Items database (i.e., the record for the state that has just been entered)

Details

One of the fields in the State database record specifies a routine to be executed when a state transition takes place. Your routine can take whatever steps are necessary to respond to the state change.

File: This is a user-written routine. It is not provided with the AdeptGEM module.

Related Routines

gm.get.sinfo
gm.get.state

Index

Index of Programs and Statements

Index of Global Variables

Adept User's Manual Comment Form

We have provided this form to allow you to make comments about this manual, to point out any mistakes you may find, or to offer suggestions about information you want to see added to the manual. We review and revise user's manuals on a regular basis, and any comments or feedback you send us will be given serious consideration. Thank you for your input.

NAME____ DATE________________________

COMPANY ____

ADDRESS____

PHONE____

MANUAL TITLE: ____

PART NUMBER and REV level:____

COMMENTS:

____

____

____

____

____

____

____

____

____

MAIL TO: Adept Technology, Inc.
Technical Publications Dept.
11133 Kenwood Rd.
Cincinnati, OH 45242

FAX: (513) 792-0274