WorkstationST Mark V GSM Server Instruction Guide

These instructions do not purport to cover all details or variations in equipment, nor to provide for every possible contingency to be met during installation, operation, and maintenance. The information is supplied for informational purposes only, and GE makes no warranty as to the accuracy of the information included herein. Changes, modifications, and/or improvements to equipment and specifications are made periodically and these changes may or may not be reflected herein. It is understood that GE may make changes, modifications, or improvements to the equipment referenced herein or to the document itself at any time. This document is intended for trained personnel familiar with the GE products referenced herein.

Public Information *– This document contains non-sensitive information approved for public disclosure.*

GE may have patents or pending patent applications covering subject matter in this document. The furnishing of this document does not provide any license whatsoever to any of these patents.

GE provides the following document and the information included therein as is and without warranty of any kind, expressed or implied, including but not limited to any implied statutory warranty of merchantability or fitness for particular purpose.

For further assistance or technical information, contact the nearest GE Sales or Service Office, or an authorized GE Sales Representative.

Revised: July 2014
Issued: May 2012

We would appreciate your feedback about our documentation.
Please send comments or suggestions to *controls.doc@ge.com*

Contents

1 Overview

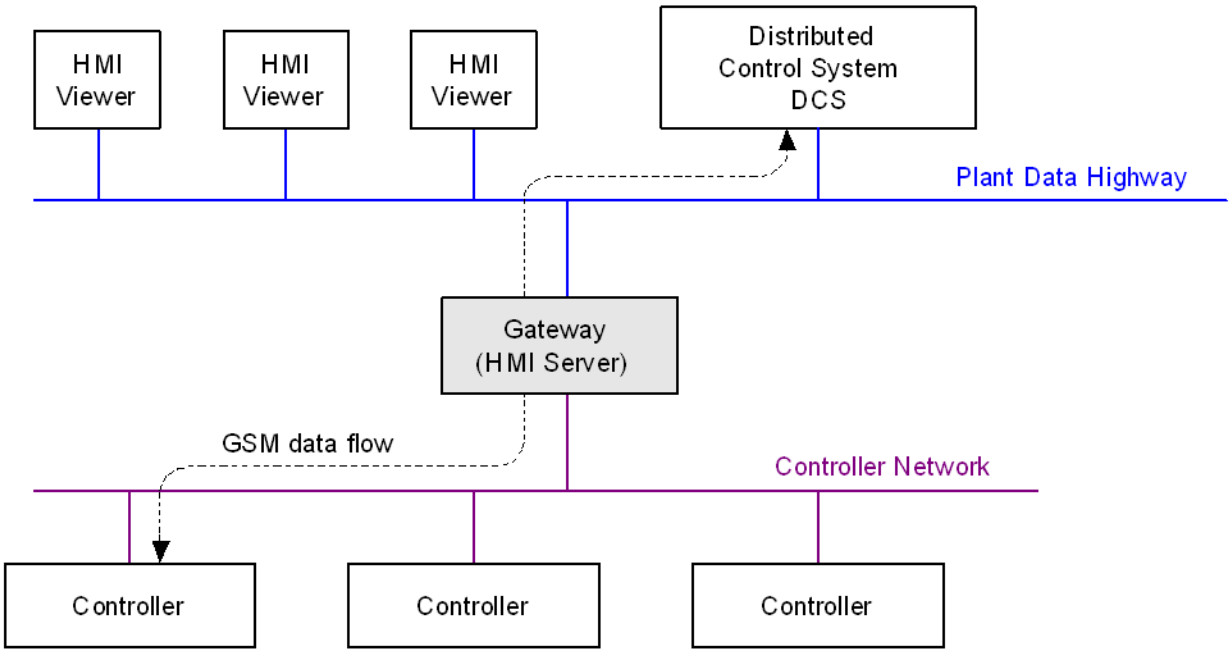
The network is typically Ethernet with TCP/IP.

This document defines message formats for communication between the turbine control system and the plant Distributed Control System (DCS). These formats are available to the DCS vendor for data collection and process control using Mark* V controllers.

The gateway is usually the Human-machine Interface (HMI) Server and may or may not be only a gateway in function.

GE Standard Messages (GSM) are application-level messages processed by software in an intervening box or gateway to the DCS. These gateways, in turn, can communicate directly with potentially several process controllers. The general function of the gateway is to act as a protocol translator yielding a consistent external interface regardless of internal protocols and data representations used. No data is emitted from the gateway unless previously requested by the DCS equipment.

GSM data flows from the controller over the data highway to the gateway and then to the DCS, and the flow of commands from the operator to the controller is as follows:



Gateway as used in a Turbine Control System

The WorkstationST* application provides two GSM servers: the WorkstationST GSM server and Mark V Feature GSM server. The WorkstationST GSM server supplies information from all controller types and provides native support for the GSM 3.0 protocol. This protocol supports Unicode characters in point names and enhanced point and alarm attributes, and emulates the GSM 1.0 and 2.0 protocols when required. A maximum of 970 points when retrieving information from a Mark V controller is a limitation shared with all other WorkstationST application real-time data functions.

The Mark V Feature GSM server supplies information from Mark V controllers only. It provides native support (no translation tables required) for the GSM 1.0 protocol, and is not limited by the data limit of 970 points. It provides the same level of support for Mark V controllers that the eTCSS generation GSM server provided.

The Mark V Feature GSM server is enabled by adding the following option to the Site Directory's CONFIG.DAT file:

```
OPTIONS
  GSM=MarkV
```

The server listens for clients using the default GSM port 768. If the WorkstationST GSM server is also in use, configure one of the GSM servers to use a different port. To configure the Mark V Feature GSM server to use a different port, supply the GSM_Port= nnn option where nnn is the port to use.

For example: To enable the Mark V Feature GSM server and configure it to use port 769, use the following CONFIG.DAT options:

```
OPTIONS
  GSM=MarkV
  GSM_PORT=
  769
```

If a port other than the default port of 768 is used, the Windows® Firewall needs to be configured to allow clients to connect using the new port. Use the default GSM port definition for port 768 as a guide and create another entry for the port selected (such as port 769 shown in the previous example).

For the previous example, the following command can be issued from an Administrator mode Command Prompt window to open a firewall entry for port 769:

```
NETSH advfirewall firewall add rule name="GE
wkstnST GSM Port2" dir=in protocol=tcp
localport=769 action=allow
```

1.1 GSM Message Types

DCS message types include:

- Administrative
- Event-driven data
- Periodic data
- Command

Administrative requests are not associated with a single process controller. They are messages from the DCS to verify gateway capability.

Sequence of events (SOE) data and alarms are available

Event-driven data messages are due to a change in state. Event-driven data includes:

- Changes in process alarm state
- Changes in digital input
- Software-detected changes in pre-defined Boolean variables
- Deadband crossings detected in pre-defined analog variables
- Changes in controller alarm (diagnostic) state

Periodic data are data sets to be transmitted back to the requester either once, or periodically at rates up to once per second. This data updates DCS status displays.

Command requests include:

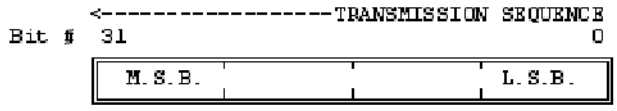
- Process alarm command requests
- Process command requests, either a momentary contact *pushbutton* or an analog setpoint.

Command requests may be rejected by the controller or blocked by the gateway.

1.2 Notation

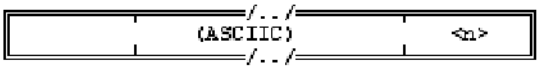
Unless otherwise noted, all data is specified in little endian format, which is the byte order for Intel® processors. For multi-byte data, the least significant byte is transmitted first while the most significant byte is transmitted last.

For notational convenience, multi-byte data is displayed in the following examples with the most significant byte to the left and the least significant byte on the right, but the transmission sequence of bytes is right to left. A 32-bit integer value is shown as follows:



The number n may be zero.

Many messages contain Name fields. The Name field allows all cases, except parameter list entries. These are counted-ASCII strings, where the first byte transmitted is treated as an unsigned number *n* followed by the name itself. The first byte following *n* is the leftmost character in the name. Name fields are shown generically as follows:

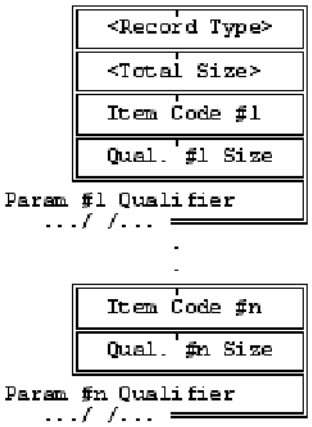


n, the record size, may be zero.

Similar to Name fields, many messages have records defined. Each record contains a Record Identifier or Type, followed by size and the record specific information. Records are shown generically as follows:

Most records within GSM messages contain groups of sub-records, which are implemented as parameter lists. Each item in a list consists of at least two 16-bit words. The first word identifies the parameter. The second word identifies the number of bytes to follow that qualify the parameter. These two words are then followed by zero or more bytes (as defined by the second word). Software may use the parameter size to skip unknown parameters.

Generic records containing parameter lists are illustrated as follows:

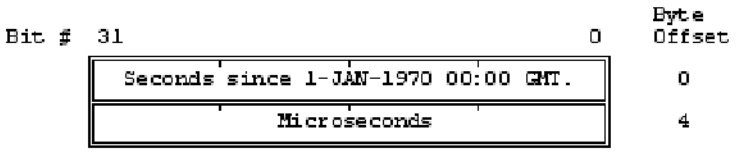


A special item code with a value of 0x0000 is the *End-of-list* item code. Even if the total record size indicates space for more item list entries, no interpretation of data beyond *End-of-list* should be attempted. Software that scans item lists must bypass unknown or previously undefined item code entries. Item Codes and Record Types have unique values. No item code has the same value as a defined record type.

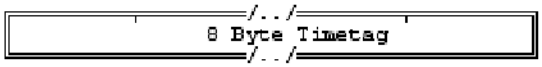
Time tag accuracy depends on the controller scan, and is less than the time tag resolution.

Most data messages contain time tagged data points. All time tags use a standard time tag format consisting of two unsigned longwords (32-bits each). The first longword is the number of seconds since 1-JAN-1970 00:00:00.000000 GMT (Greenwich Mean Time). The second longword is the number of microseconds within the second, having a decimal range of 0-999,999. The latest date and time that can be stored is 07-FEB-2106 06:28:15.999999 GMT.

The layout of a time tag is as follows:



Within a message format specification, the time tag is as follows:

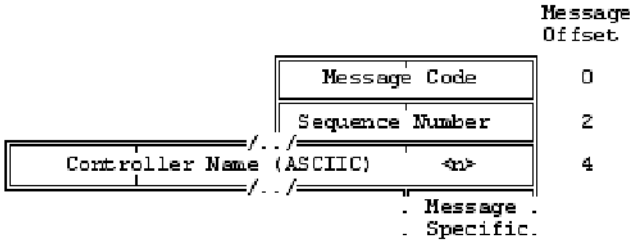


Many of the message formats have RESERVED fields for future GSM versions. To preserve backward/forward compatibility, software that generates GSM messages inserts zeros in these fields. Software that receives GSM messages must ignore these fields.

GSM allows data acquisition and control of multiple process controllers. The generic GSM messages contain a message header consisting of three parts:

- A 16-bit message code that identifies the requested function
- An arbitrary 16-bit sequence number generated by the data/command requester that identifies a request; all responses to a given request message have this sequence number echoed back allowing stale data to be thrown away.
- An ASCIIC process controller name that defines the controller receiving the request.

A GSM message has a generic form as follows:



1.2.1 Time Tag Considerations

As previously shown, the resolution of the standard time tag is one microsecond. All time tagged data is stamped at the source of the data, using the resolution available in the local process controller. The resolution of time stamps in the local process controller depends on the frame rate (control execution rate).

The Mark V controller time-tags SOE data to within 1 ms.

Time tagging considerations include the I/O scan rates used. For example, certain control functions may only scan inputs at an eight Hz rate. This means that consecutive data samples collected are time tagged in multiples of 125 milliseconds. The Mark V controller typically runs at 16 Hz, and periodic data, alarms, and events are time tagged at this frame rate. Discrete Sequence of Events data is tagged faster than this.

Time tag accuracy or coherency across different process controllers is the subject of time setting and time synchronization. GSM does not define the capability of time setting using messages over a network, nor does it allow the concept of time synchronizing process controllers over a network. Where required, time setting and time synching of process controllers can be handled by external equipment utilizing a common time reference. Time tag coherency in these cases is achieved to an accuracy as demanded by system requirements.

High-resolution time tagged data does not necessarily imply speedy reporting of data. Some process controllers, for example, may buffer multiple pieces of time tagged data into envelopes, which are delivered at a later time. Due to the asynchronous nature of delivering buffered time-tagged data, it is both possible and likely that different pieces of data will not be delivered in chronological order.

1.3 Summary of Messages

A summary of the GSM message types and associated parameters are as follows:

Message Code Summary

Message Code	Message Type	Sent By
0x0100	Supported Controller Request	DCS
0x0101	Supported Controller Response	Gateway
0x0200	Heartbeat Message	DCS
0x0300	Alarm Establish Request	DCS
0x0301	List Establish ACK/NAK	Gateway
0x0302	Alarm Data Message	Gateway
0x0400	Digital Input Record Establish Request	DCS
0x0402	Digital Input Record Data Message	Gateway
0x0500	Software Event Record Establish Request	DCS
0x0502	Software Event Data Message	Gateway
0x0600	Periodic Data Request	DCS
0x0601	Periodic Data ACK/NAK Response	Gateway
0x0602	Periodic Data Message	Gateway
0x0700	Alarm Command Request	DCS
0x0701	Alarm Command ACK/NAK	Gateway
0x0702	Alarm Dump Message	Gateway
0x0800	Process Control Command Request	DCS
0x0801	Process Control ACK/NAK Response	Gateway

Record Identification Summary

Record ID	Record Type
0x8000	List of sub-records
0x8100	Controller Information parameter list
0x8300	Alarm Information parameter list
0x8400	Digital Input Information parameter list
0x8500	Software Event Information parameter list

Parameter Identification Summary

Param ID	Parameter Type
0x0000	End of parameter list
0x1000	Process controller name
0x1010	Number of communication links to process controller
0x1020	Process controller type
0x1030	Point name (short name)
0x1040	Time tag
0x1050	Alarm drop number
0x1060	Point value
0x1070	Alarm locked state
0x1080	Alarm reason code
0x1090	Point text (long name)
0x10A0	Alarm sequence number
0x10B0	Process control setpoint value
0x10C0	Alarm ACK state
0x10D0	Point Identifier Hint
0x10E0	Point Status List
0x10F0	Point Length Report

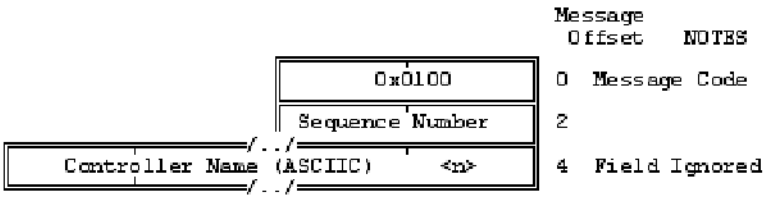
2 Administrative Requests

This section defines messages from the DCS to the gateway that request the identity of the connected controllers, and periodically inform the gateway that the DCS is functioning.

2.1 Supported Controller Request

The DCS issues a supported controller request to the gateway to determine which process controllers the gateway can communicate with, along with the current communication status. This request is a DCS-to-gateway message, and as such the *Controller Name* field is inappropriate. The gateway ignores this field, but the DCS should still insert a zero length name.

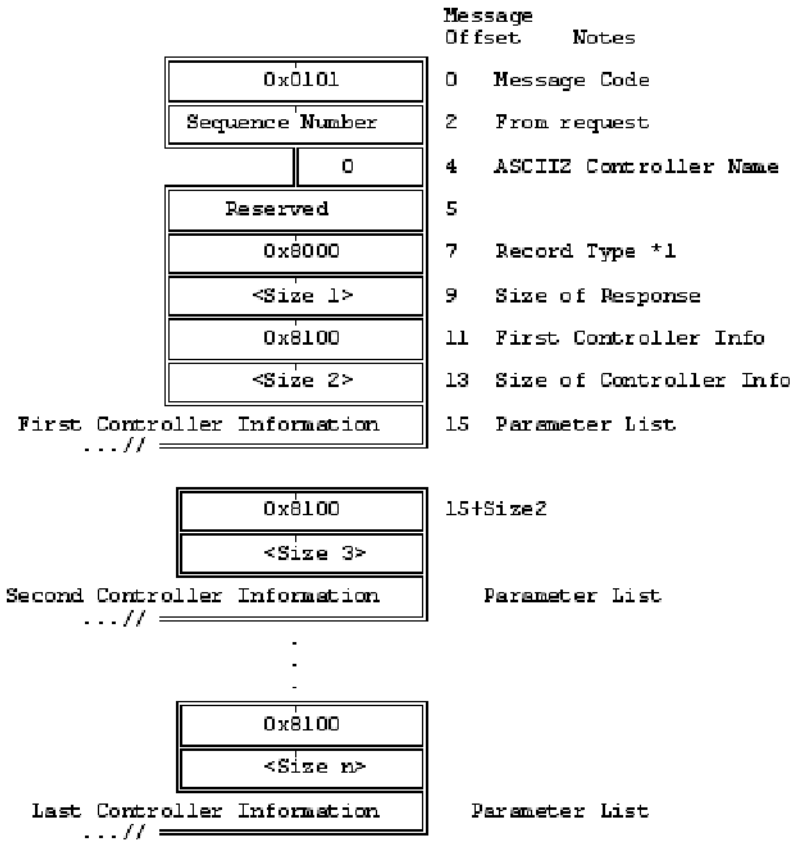
The message format is:



2.1.1 Supported Controller Response

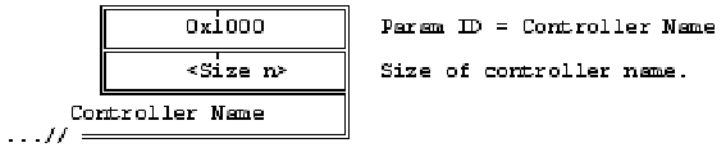
The gateway responds to the request with a list of supported controllers, controller type, and current communication status. The response is a series of parameter lists where each list defines information about a single controller.

The supported controller response format is:

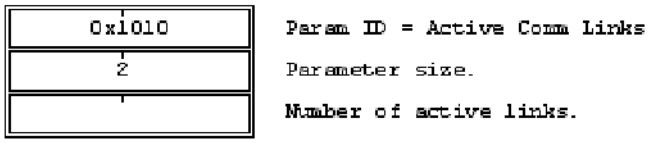


This response contains a list of parameter lists (denoted by *1) identified by a record type value of 0x8000. Each parameter list corresponds to a supported controller. Possible parameters are defined here.

The controller name is defined as follows:

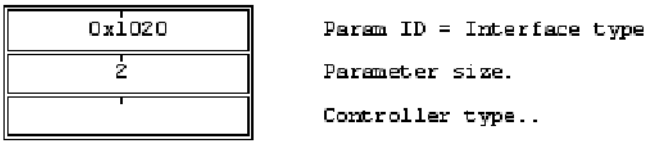


The number of communication links currently active between the gateway and the controller is defined as follows:



A value of zero indicates that the gateway supports the controller, but cannot currently communicate with it.

The interface type is defined as follows:

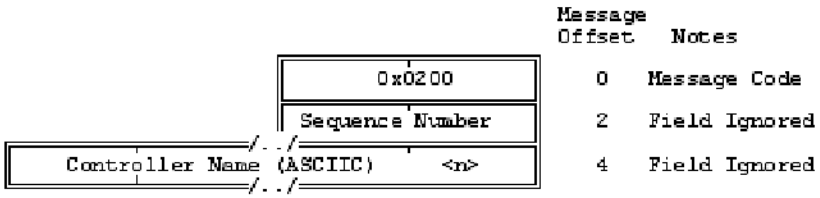


The currently defined interface type is 0001 - Turbine Controller. Other interface types may be defined in the future.

2.2 Heartbeat Message

The DCS requests to be on various distribution lists defined in the sections *Administrative Requests* and *Event-driven Requests*. The Heartbeat message is a status message in a DCS-to-gateway administrative message that should be transmitted approximately every 20 seconds. If a heartbeat message is not received for a period of 60 seconds, any DCS data lists are automatically canceled and connection to the DCS is terminated. The gateway does not respond to heartbeat messages.

The Heartbeat message format is:



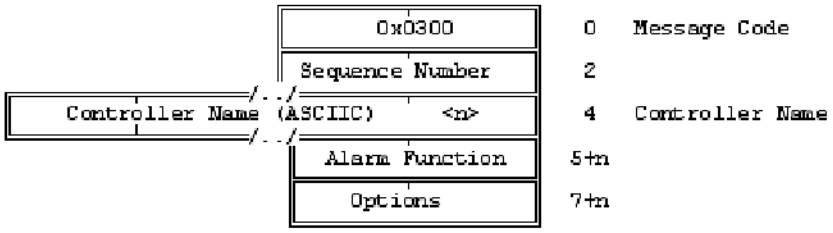
3 Event-driven Requests

This section describes alarm handling, discrete SOE data, and communication between the controller and the DCS.

3.1 Alarm Record Establish Request

This message requests that the gateway place the DCS on, or remove it from, the distribution list for process alarm messages. Process alarm messages defined below are container messages issued spontaneously any time a process alarm changes state. State changes include transitions in alarm value, alarm lock/unlock transitions, and removal of an alarm from the process controller's alarm queue (if supported in the controller).

The alarm establish request format is:



Alarm Function values are:

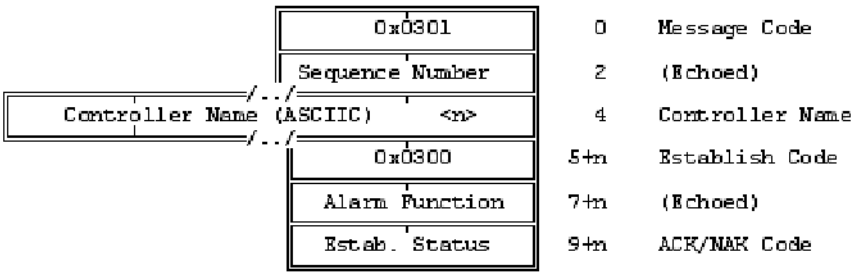
- 0x0000 requests that the DCS be added to the alarm distribution list for the specified controller.
- 0xFFFF requests that the DCS be removed from the alarm distribution list.

All other values are reserved.

Options request that other information be included in spontaneously sent alarm messages. Bit 0 = 1 requests that alarm text be included in alarm messages; otherwise alarm text will be omitted.
Bits 1-15 are reserved and must be zero.

3.1.1 Alarm Record Establish ACK/NAK Response

The gateway responds to an alarm record establish request with an establish ACK/NAK response. The establish ACK/NAK response format is:



The Sequence Number, Controller Name, and Alarm Function are echoed from the alarm establish request. The establish code is 0x0300 for an alarm ACK/NAK response.

Alarm Record Establish ACK/NAK Response Codes

Code	Description
0	Success
+1	DCS is on the gateway's distribution list. Communication with the process controller is not currently possible
-1	Unknown controller name
-2	Function not supported by process controller
-3	Gateway distribution list table is full
-4	Malformed request

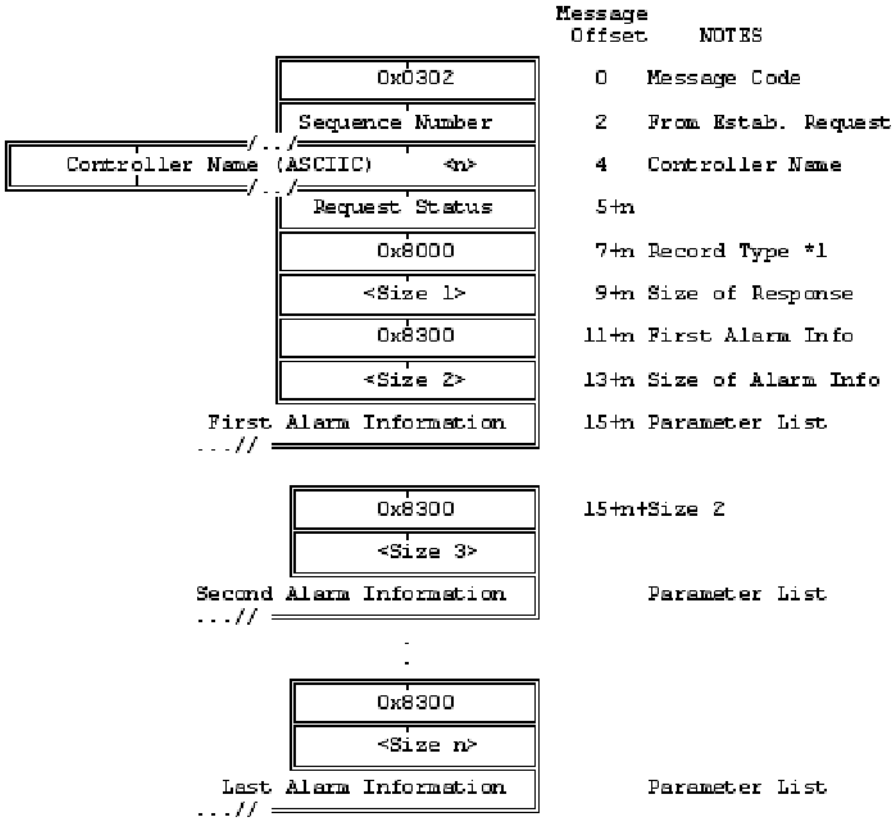
3.2 Alarm Data Messages

Alarm data messages are sent spontaneously to any DCS on the gateway's distribution list for alarm messages. Alarm data messages are container messages with one or more alarms that have changed state. These messages do not provide the current status of all alarms.

The Mark V controller alarm queue can contain up to 64 entries. All Mark V controller alarm points are available as periodic data.

The alarm data message contains parameter lists that define information about a single alarm. Not all controllers support all possible parameters and information not relevant to a given controller is not included.

The alarm data message is:

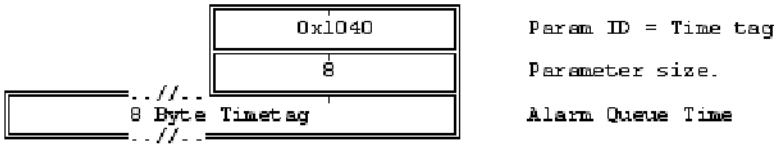


These messages contain a list of parameter lists (denoted by *1) identified by a record type value of 0x8000. Each parameter list corresponds to a single alarm that has changed state. The possible parameters are defined here.

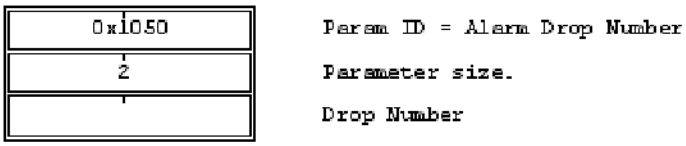
The short name format of the alarm is:

This parameter is not included if the gateway cannot translate the alarm drop into its short name form.

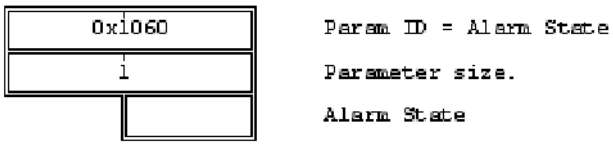
The time tag format associated with the alarm record is:



The alarm drop number format is:

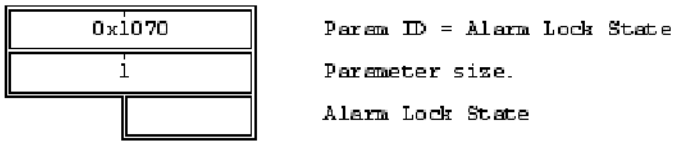


The alarm state format is:



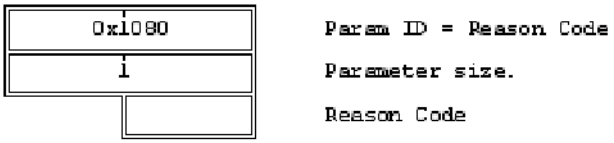
Bit 0 defines the current alarm state
Bit 0 = 0 if out of alarm
Bit 0 = 1 if in alarm
All other bits are reserved.

The alarm lock state format is:



Bit 0 defines the current alarm lock state
Bit 0 = 0 if alarm is not locked
Bit 0 = 1 if alarm is locked
All other bits are reserved.

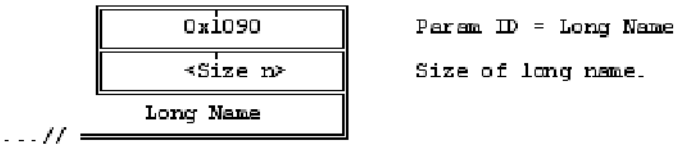
The reason code parameter defines why the alarm record is being sent. It is used for remote alarm queue management.



Alarm Data Reason Codes

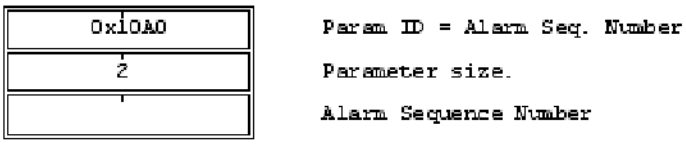
Code	Description
0x00	Not used
0x01	Alarm state just transitioned
0x02	Alarm just locked
0x03	Alarm just unlocked
0x04	Reserved
0x05	Alarm just re-triggered. Reset time tag in the alarm queue
0x06	Reserved
0x07	Alarm just acknowledged
0x08	Alarm reset. Remove alarm from alarm queue
0x09	Alarm Dump Record
0xFE	End of Alarm Dump
0xFF	Clear Alarm Queue to prepare for alarm dump
All other values are reserved	

The long name (text) format is:



This information is not sent if it is not requested in the alarm establish message. The text size is zero if the gateway cannot provide the associated text.

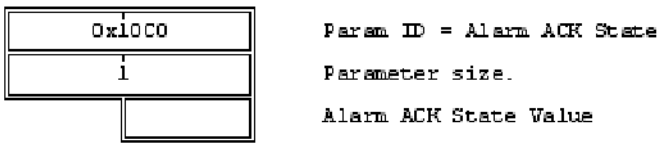
The alarm record sequence number format is:



Refer to the section Command Messages for the alarm dump message.

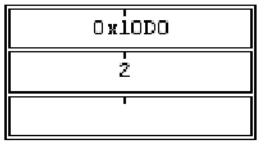
Each alarm queue has a sequence number, which incrementes for every alarm record. The alarm sequence number is provided for remote alarm queue management. If the remote alarm queue sequence number (plus 1) does not equal this sequence number, the remote alarm queue must be re-synchronized with the process controller's alarm queue. (The DCS should request an alarm dump to re-synchronize.) This value is a 2-byte unsigned integer that will rollover after 0xFFFF.

The alarm ACK state parameter is:



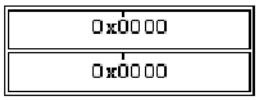
Bit 0 defines the acknowledged alarm state
Bit 0 = 0 if the alarm has not been acknowledged
Bit 0 = 1 if the alarm has been acknowledged
All other bits are reserved.

The point ID hint parameter is:



The gateway sends this parameter if it cannot translate the alarm drop number into its short name form. This parameter indicates incomplete translation tables in the gateway.

The end of the list parameter is:

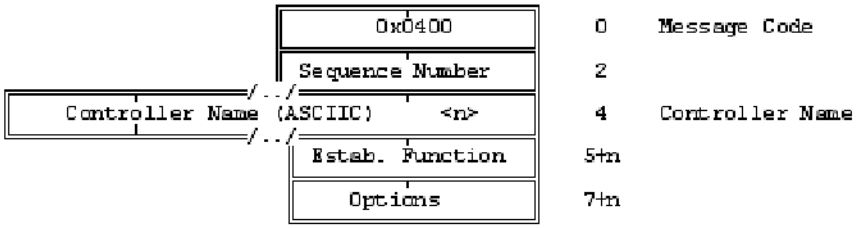


No more information about this alarm follows.

3.2.1 Digital Input Record Establish Request

This message requests that the gateway place the DCS on, or remove it from, the distribution list for digital input messages. Digital input messages are container messages issued spontaneously when a digital input in the process controller changes state (if supported in the controller). The gateway always responds to this request with an establish ACK/NAK message.

The digital input establish request format is:



Establish Function has two values:

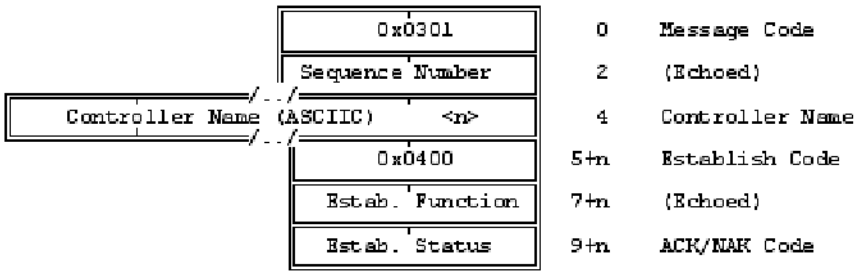
- 0x0000 requests that the DCS be added to the digital input distribution list for the specified controller.
- 0xFFFF requests that the DCS be removed from the distribution list. All other values are reserved.

Options request other information be included in spontaneously sent digital input messages.
Bit 0 = 1 requests that longname descriptive text be included in any generated digital input messages; otherwise longname text will be omitted.
Bits 1-15 are reserved and must be zero.

3.2.2 Digital Input Record Establish ACK/NAK Response

The gateway responds to a digital input establish request with an establish ACK/NAK response.

The establish ACK/NAK response format is:



The Sequence Number, Controller Name, and Establish Function are echoed from the digital input establish request. The establish code = 0x0400 for a digital input ACK/NAK response.

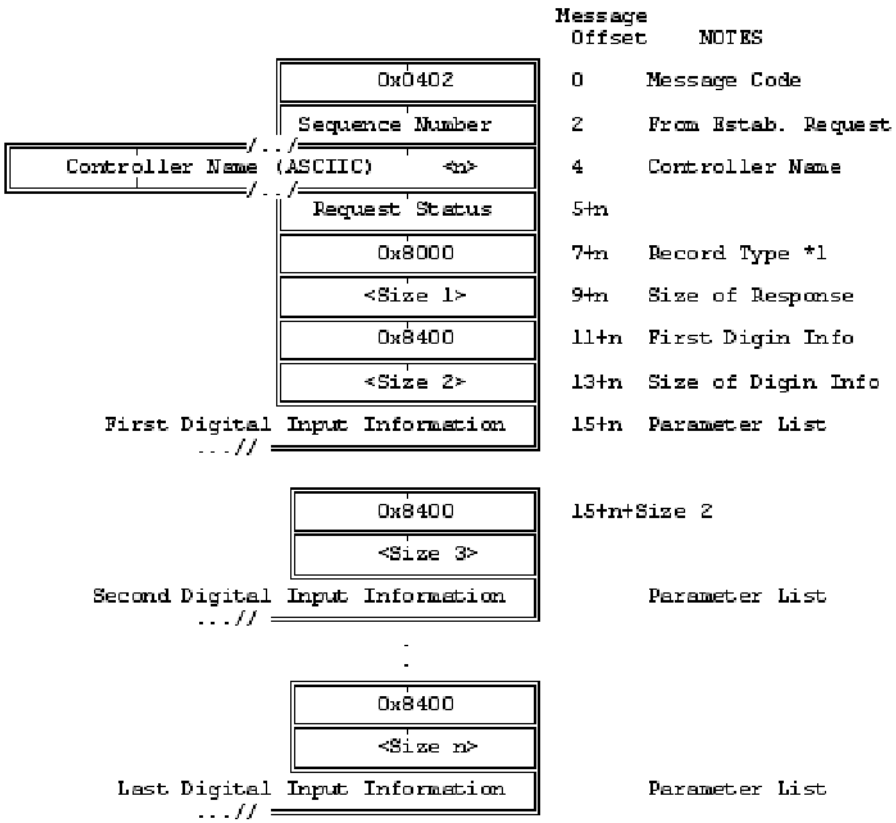
Digital Input Record ACK/NAK

Code	Description
0	Success
+1	DCS is on the Gateway's distribution list. Communication with the process controller is not currently possible
-1	Unknown Controller Name
-2	Function not supported by process controller
-3	Gateway distribution list table is full
-4	Malformed request

3.2.3 Digital Input Data Messages

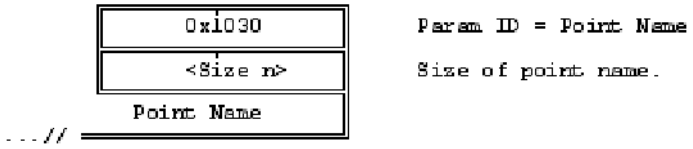
Digital input data messages are sent spontaneously to any DCS on the gateway's distribution list for digital input messages. Digital input data messages are container messages with one or more inputs that have changed state, such as SOE data.

This message contains parameter lists that define information about a single digital input. Not all process controllers support all possible parameters and information not relevant to a given process controller is not included.



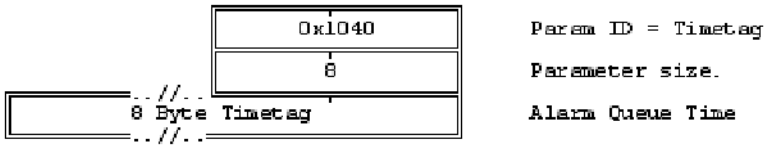
These messages contain a list of parameter lists (denoted by *1) identified by a record type value of 0x8000. Each parameter list corresponds to a single digital input, which has changed state. Possible parameters are shown here.

The point name parameter is:

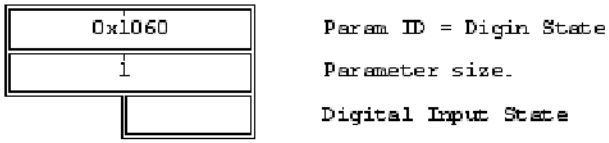


This parameter defines the short name (point name) of the digital input. This parameter is not included if the gateway is incapable of translating the digital input number into its short name form.

The time tag parameter is:

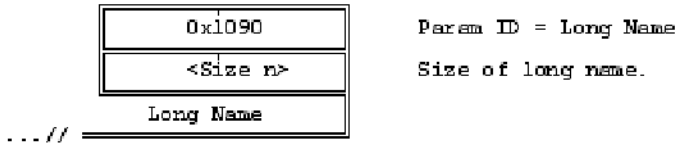


The time digital input changed state parameter is:



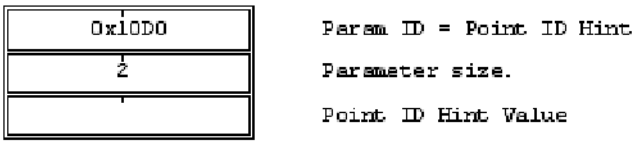
Bit 0 defines the current state; all other bits are reserved.

The long name (text) associated with the digital input is:



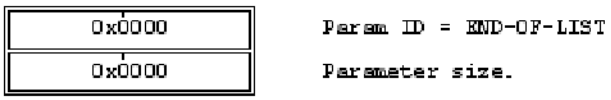
This information is not sent if it was not requested in the digital input establish message. Text size is zero if the gateway cannot provide the associated text.

The point ID hint parameter is:



The gateway sends this parameter if it cannot translate the digital input number into its short name form. This parameter indicates incomplete translation tables in the gateway.

The end of list parameter is:

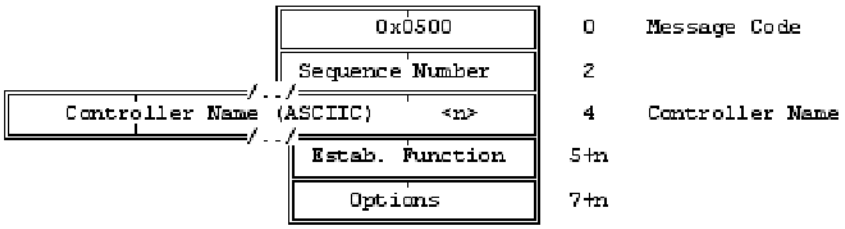


This indicates there is no more information about this digital input.

3.2.4 Software Event Record Establish Request

This message requests that the gateway place the DCS on, or remove it from, the distribution list for software event messages. Software events are usually caused by a step in a sequence or a change in the status of a device, such as breaker position. Event messages defined here are container messages issued spontaneously any time a logic variable in the controller changes state (if supported in the controller). The logic variables to be change-detected by software are predefined in the controller. The gateway always responds to this request with an establish ACK/NAK message.

The software generated event establish request format is:



Establish Function values include:

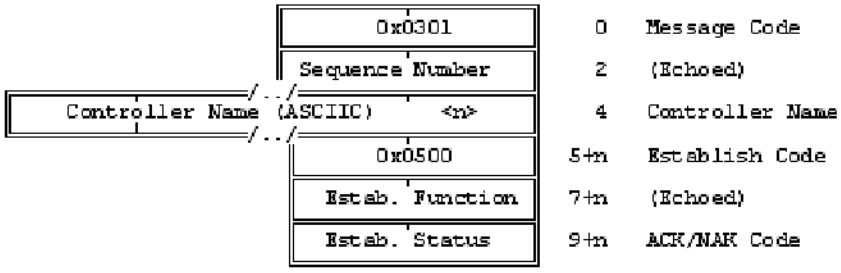
- 0x0000 requests that the DCS be added to the event message distribution list for the specified controller.
- 0xFFFF requests that the DCS be removed from the distribution list. All other values are reserved.

Options request other information be included in spontaneously sent event messages.
Bit 0 = 1 requests that longname descriptive text be included in any generated software event messages; otherwise, longname text is omitted.
Bits 1-15 are reserved and must be zero.

3.2.5 Software Event Record Establish ACK/NAK Response

The gateway responds to an event establish request with an establish ACK/NAK response.

The establish ACK/NAK response format is:



The Sequence Number, Controller Name, and Establish Function are echoed from the event establish request. The establish code is 0x0500 for an event ACK/NAK response.

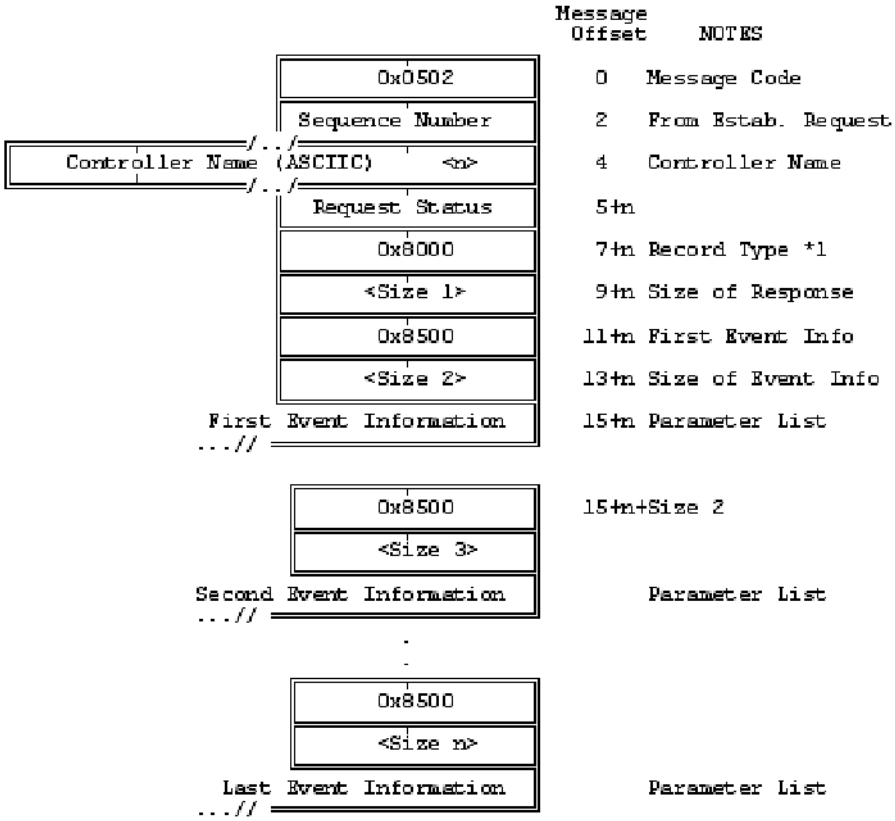
Software Event Record Establish ACK/NAK Codes

Code	Description
0	Success
+1	DCS is on the gateway's distribution list. Communication with the process controller is not currently possible.
-1	Unknown controller name
-2	Function not supported by the process controller
-3	Gateway distribution list table is full
-4	Malformed request

3.2.6 Software Event Data Messages

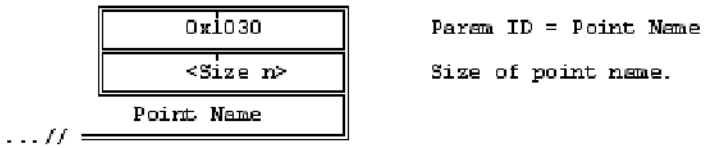
Software event data messages are sent spontaneously to any DCS on the gateway's distribution list for event messages. Event data messages are container messages containing one or more logic points, which have changed state.

This message contains parameter lists that define information about a single logic variable. Not all controllers support all possible parameters and information not relevant to a given controller is not included.



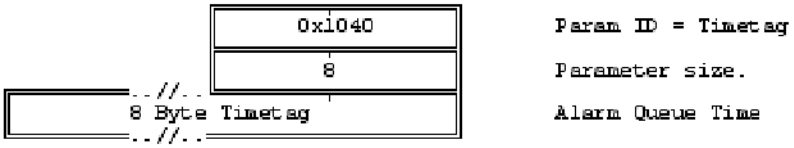
These messages contain a list of parameter lists (denoted by *1) identified by a record type value of 0x8000. Each parameter list corresponds to a single logic value, which has changed state. Possible parameters are shown here.

The short name of the event is:

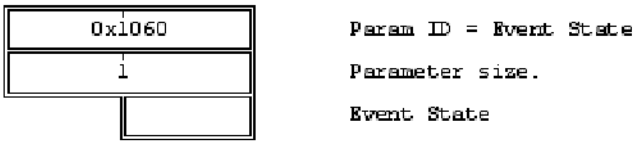


This parameter is not included if the gateway cannot translate the event number into its short name form.

The time logic variable changed state parameter is:

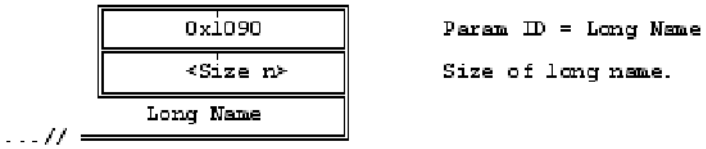


The event state parameter is:



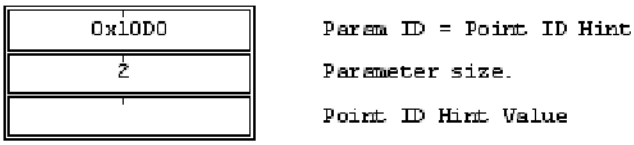
Bit 0 below defines the current state of the logic variable. All other bits are reserved.

The long name (text) associated with the event is:



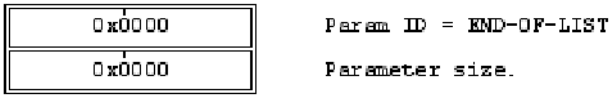
This information is not sent if it was not requested in the software event establish message. Text size is zero if the gateway cannot provide the associated text.

The point ID hint parameter is:



The gateway sends this parameter if it cannot translate the event number into its short name form. This parameter indicates incomplete translation tables in the gateway.

The end of the list parameter is:



No more information about this event follows.

4 Periodic Data Messages

This section describes how data is requested and sent to the DCS on a continuing basis. The data request, acknowledgement, and data messages are also defined.

4.1 Periodic Data Request

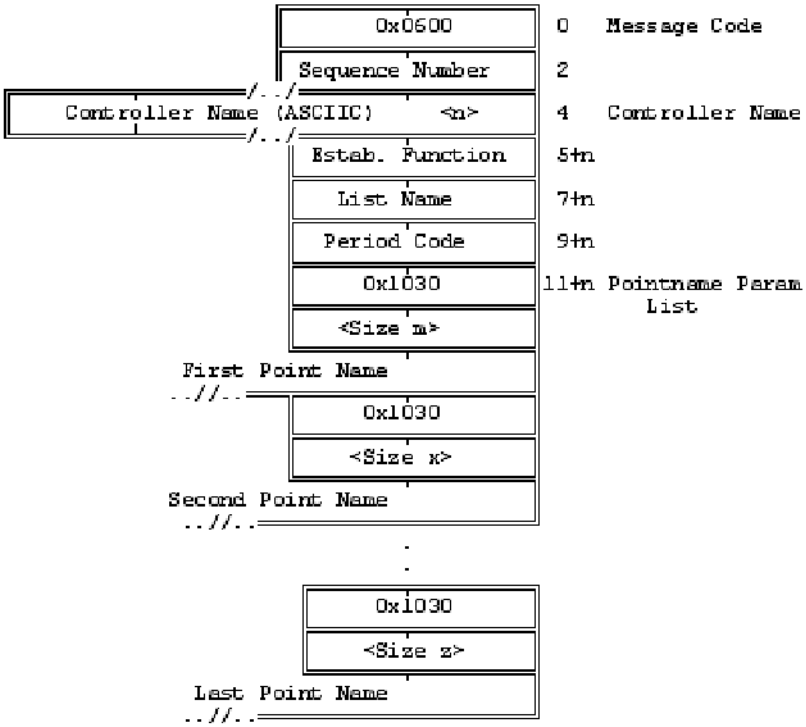
The DCS operator interface requires data on a regular basis (periodic data).

The DCS can define lists of data to be returned on a periodic basis. This is typically used to maintain real-time databases for display purposes. Using this method, any data point defined in the process controller (and also defined in the gateway's database) can be returned to the DCS. Multiple data lists may be defined by the DCS. Limitations are placed on list definitions based on the controller as follows:

Controller	Maximum Points per List	Max Number of Lists
Mark V	96	32

The gateway responds to a periodic data request with a periodic data ACK/NAK message.

The periodic data list definition is:



List Name combined with Controller Name defines a unique set of data points to be returned. If the request contains a List Name/Controller Name identical to a previously defined set, the new request supersedes the previous definition. List Name may be any value.

Establish Function has two values:

- 0x0000 requests that data points specified in the request be returned to the DCS according to the Period Code.
- 0xFFFF requests cancellation of any previously defined list corresponding to the List Name/Controller Name.

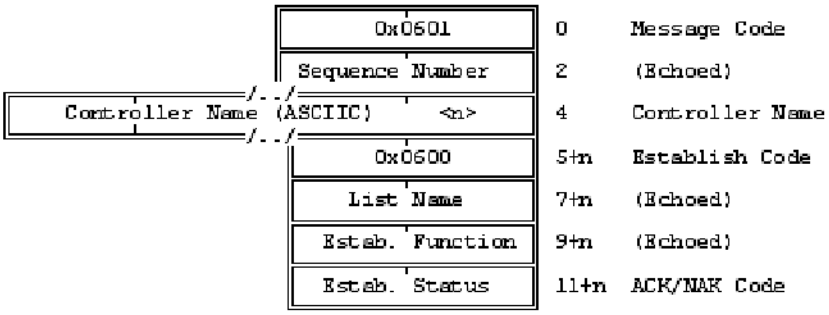
Period Code defines the number of seconds between data transmissions. It is ignored on list cancellation requests. A Period Code with a value of zero requests the data to be transmitted only once.

The Point Name parameter list is ignored on list cancellations resulting from the Establish Function. The Point Name parameter is an ASCII string, and must match the point name defined in the controller configuration.

4.2 Periodic Data ACK/NAK Response

The gateway responds to a periodic data request with a periodic data ACK/NAK response.

The periodic data ACK/NAK response is:



The Sequence Number, Controller Name, List Name, and Establish Function are echoed from the event establish request. The establish code is 0x0600 for a periodic data ACK/NAK response.

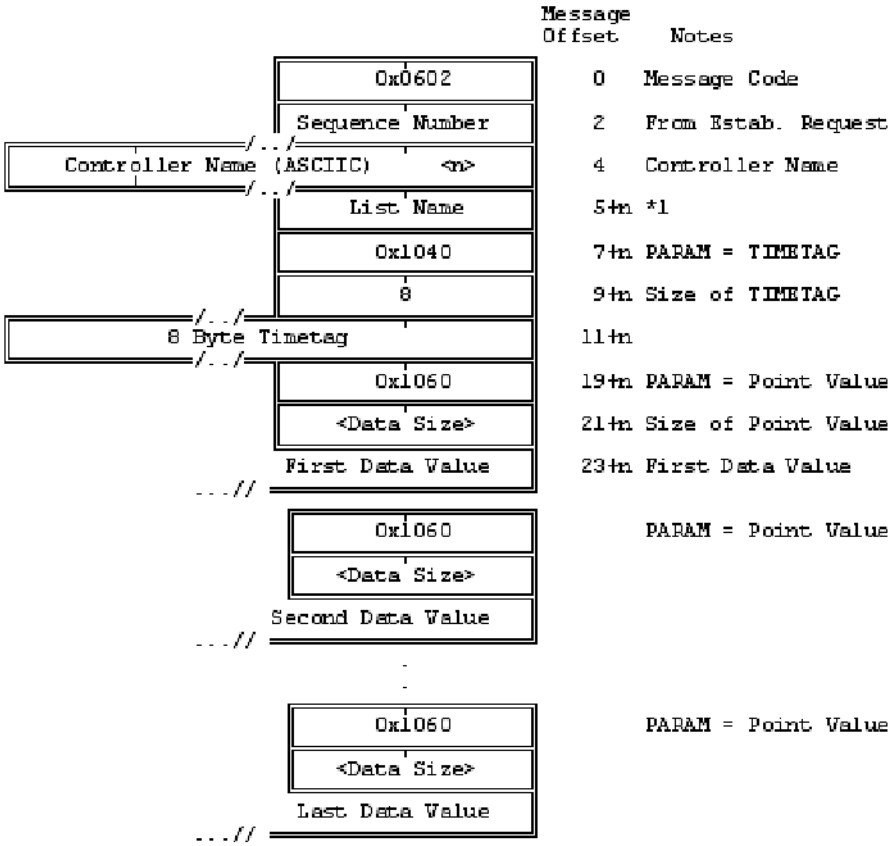
Periodic Data ACK/NAK Codes

Codes	Description
0	Success
+1	DCS on gateway's distribution list. Communication with the process controller is not currently possible
-1	Unknown controller name
-2	Function not supported by the process controller
-3	Gateway periodic list definition table is full
-4	Malformed request
-5	Internal gateway error
-6	All points requested are undefined
-7	Too many data points defined in request

4.3 Periodic Data Message

Periodic data messages are sent to the DCS at the rate defined in the period code, following the transmission of the periodic data ACK/NAK message. These messages consist of a small header identifying which data list is contained in the message, followed by a parameter list containing a time tag and point values. The point value list is transmitted in the same order as defined in the periodic data request. If a requested point is undefined, the returned list entry shows a size of zero.

The periodic data message format is:



*1 – The *List Name* determines which set of data values are being returned if the DCS has defined multiple lists.

Note If the gateway loses communication with the process controller, these messages stop. Message transmission resumes automatically when communication is re-established.

4.4 Sources of Periodic Data

The HMI provides data for periodic data requests from various subsystems. The subsystem used depends upon controller type.

Each subsystem has its own rules for the point name format and the scaling of raw data returned.

Example: The current turbine speed may be requested from a Mark V turbine controller. When requesting the point directly, the controller name is T1, the point name is TNH, and the result is a 2-byte fixed point integer.

The following defines which subsystem is used to satisfy a periodic data request based upon the controller type (determined from the controller name):

Mark V Direct Data obtained by direct communication with the Mark V controller.

Overview This subsystem works by sending a periodic data list definition to the Mark V controller which matches the periodic data list sent to the GSM server. The Mark V controller messages are limited to 256 bytes each, which translates into 96 points maximum for a Mark V controller. When the Mark V controller sends a data message to the GSM server it reformats it into a GSM message and forwards it to the DCS.
If the messages stop from the Mark V controller (such as when the Mark V controller is powered down or the communication cable is disconnected) then the messages will stop to the DCS.

Controller Name The name of the Mark V controller. The unit names can be verified in the HMI in the CONFIG.DAT file in the Site Directory.

Point Name The signal name in the Mark V controller.

Time Tag The time in the GSM message the Mark V controller returned along with the data.

Data Types The Mark V controller returns the data in its internal data format. Logic signals are returned in a byte (including the forcing bit) and analog signals are returned as 16-bit integers. Each point has its own scale code that determines the gain and offset to use to convert the 16-bit value to engineering units.

Limitations Each data list is limited to 96 points per list. The GSM server will not allow more than 32 lists to be defined for any one Mark V controller.

The Mark V controller can typically supply a maximum of 48 data lists. These lists are used by each HMI, Historian, and OSM to collect data, with each supervisory computer limited to no more than ten (10) data lists for its normal background data retrieval. Depending upon the number of computers requesting Mark V controller data and the number of lists each one is requesting, the Mark V controller can run out of data lists. The GSM server does not have access to the number of lists currently defined in the Mark V controller, tries to collect all the data lists that the DCS has defined even if that causes the total number of data lists in the Mark V controller to be exceeded.

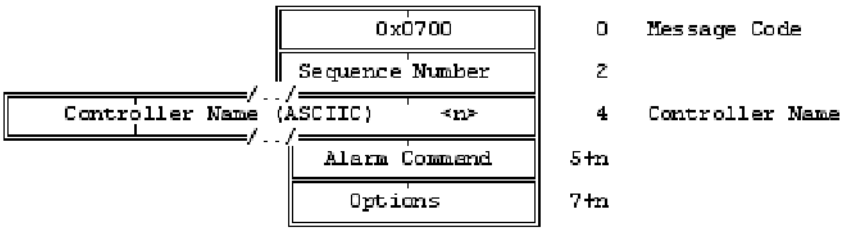
5 Command Messages

This section describes the operator messages used to acknowledge, reset, lock, or unlock alarms in the controller, and to request an alarm dump. Messages to initiate control actions such as start/stop and change setpoint are covered.

5.1 Alarm Command Request

Alarm command requests are used for remote control of the turbine controller's alarm queue.

The alarm command request format is:



Options qualify the alarm command. The available alarm command values and options are as follows:

Command Value	Command Function	Options Meaning
2	Lock Single Alarm	Alarm Drop (index) Number
3	Unlock Single Alarm	Alarm Drop (index) Number
4	Acknowledge Alarms	Number of alarms *1
6	Reset All Alarms	<Ignored>
7	Acknowledge 1 Alarm	Alarm Drop Number
8	Reset 1 Alarm	Alarm Drop Number
10	Alarm Silence	<Ignored>;
255	Request Alarm Dump	Include Alarm Text Flag *2
All other values for alarm command are reserved.		

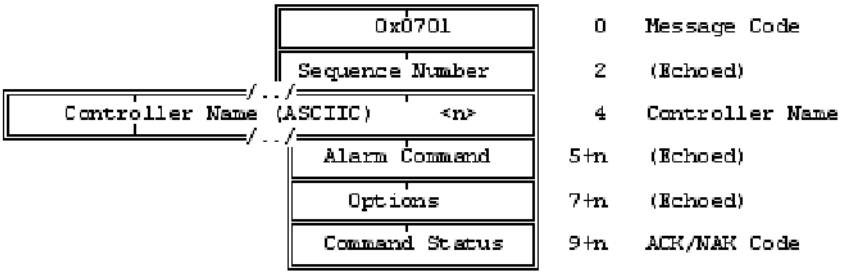
*1 – An options value of 0xFFFF acknowledges all alarms, otherwise a maximum of 12 alarms can be acknowledged.

*2 – Options bit 0 = 1 requests the alarm dump include alarm text, otherwise the alarm text is omitted. Bits 1-15 are reserved.

5.1.1 Alarm Command ACK/NAK Response

The gateway responds to an alarm command request with an ACK/NAK response. This informs the DCS that the command is valid, and has been sent successfully. To confirm that the controller has the command, look for the associated alarm queue change message.

The alarm command ACK/NAK response format is:



The Sequence Number, Controller Name, Alarm Command, and Options are echoed from the alarm command request.

Alarm Command ACK/NAK Codes

Codes	Description
0	Success. The alarm command sent to process controller
+1	Request received by the gateway, but communication with the process controller is not currently possible
-1	Unknown controller name
-2	Function not supported by process controller
-3	Invalid command
-4	Malformed request
-5	Internal Gateway error
-6	Permission violation due to control hierarchy

5.2 Alarm Dump Messages

All current alarms can be sent to the DCS.

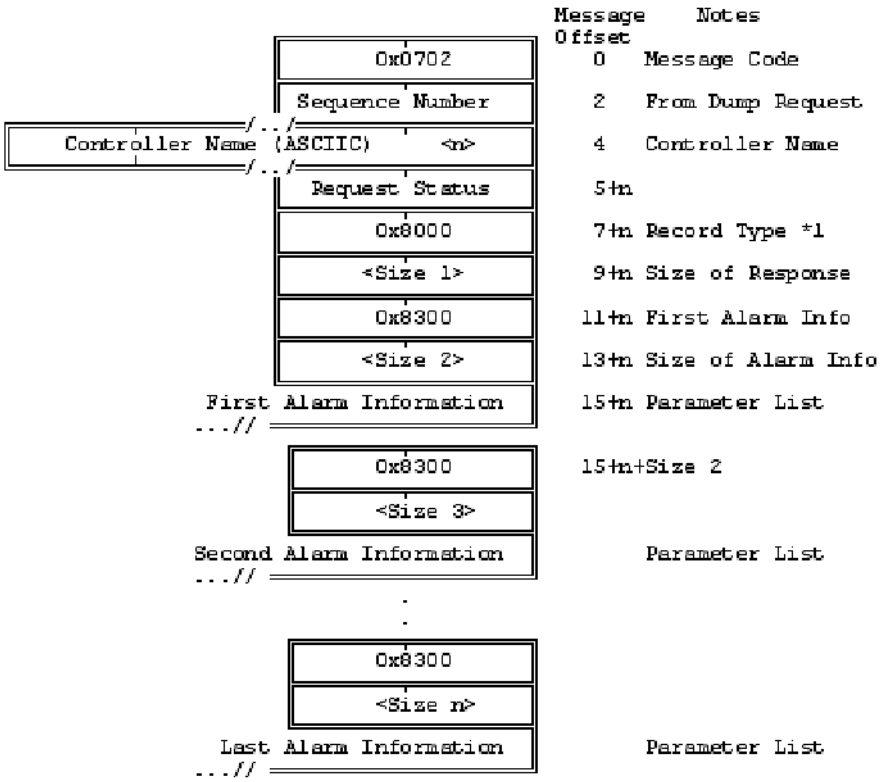
Alarm dump messages are nearly identical to alarm data messages. They are sent following the alarm command ACK/NAK message, if an alarm dump was requested and supported by the controller. Mark IV, V, and VI controllers support this function. Alarm dump messages provide the current status of all alarms. Due to the potential size of the entire alarm queue information, this message may be broken up into several GSM messages.

The alarm sequence number in an alarm dump is the latest alarm sequence number used in the controller. This same number is used for multiple GSM messages. The DCS can use the alarm sequence number to synchronize its own alarm queue.

5.2.1 Alarm Dump Message Structure

The message contains parameter lists that define information about a single alarm. Not all controllers support all possible parameters and information not relevant to a given controller is not included.

The alarm dump message structure is:

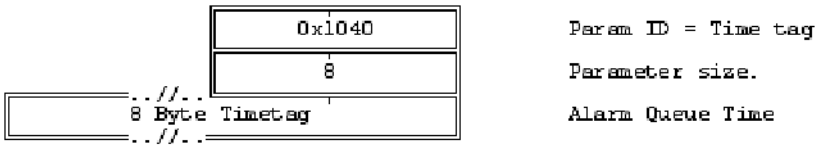


These messages contain a list of parameter lists (denoted by *1) identified by a record type value of 0x8000. Each parameter list corresponds to a single alarm in the queue. If the alarm queue is empty, there will be one parameter list containing only the End-Of-List parameter.

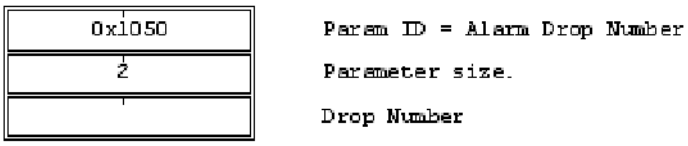
The alarm short name is defined by the parameter as follows:

This is not included if the gateway is incapable of translating the alarm drop into its short name form.

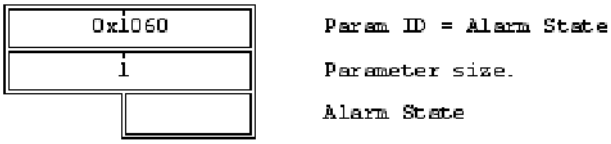
The time tag associated with the alarm record is:



The alarm drop number parameter is:

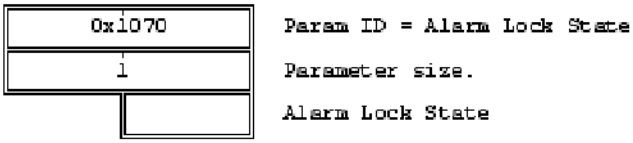


The alarm state parameter is:



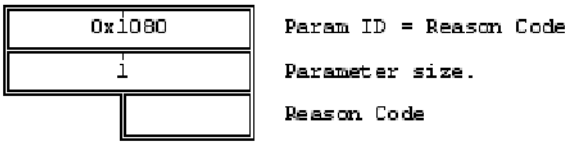
Bit 0 defines the current alarm state.
Bit 0 = 0 if out of alarm
Bit 0 = 1 if in alarm. All other bits are reserved.

The alarm lock state parameter is:



Bit 0 defines the current alarm state.
Bit 0 = 0 if alarm is not locked
Bit 0 = 1 if alarm is locked. All other bits are reserved.

The alarm reason code parameter is:

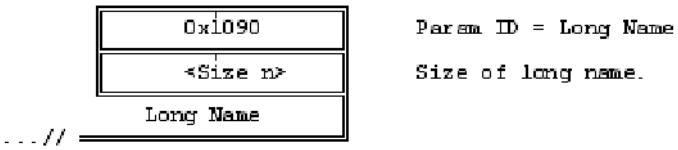


This defines why the alarm record is being sent, and is used for remote alarm queue management.

Alarm Dump Reason Codes

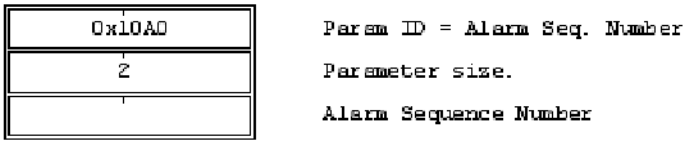
Codes	Description
0x00	Not used
0x01	Alarm state just transitioned
0x02	Alarm just locked
0x03	Alarm just unlocked
0x04	Reserved
0x05	Alarm just re-triggered. Reset time tag in alarm queue
0x06	Reserved
0x07	Alarm just acknowledged
0x08	Alarm reset. Remove alarm from alarm queue
0x09	Alarm Dump Record
0xFE	End of Alarm Dump
0xFF	Clear Alarm Queue to prepare for alarm dump.
All other values are reserved.	

The long name (text) associated with the alarm is:



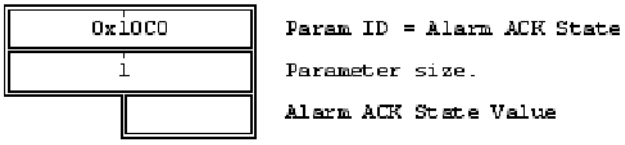
This information is not sent if it was not requested in the alarm establish message. The text size will be zero if the gateway cannot provide the associated text.

The alarm record sequence number parameter is:



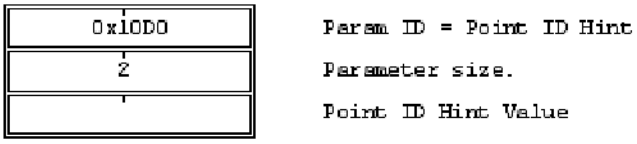
Each alarm queue has a sequence number that increments for every alarm record. If the remote alarm queue sequence number (plus 1) does not equal this sequence number, the remote alarm queue must be re-synchronized with the process controller's alarm queue. (The DCS should request an alarm dump to re-synchronize). The alarm sequence number is a 2-byte unsigned integer that will rollover after 0xFFFF.

The Acknowledged State of the alarm is identified by the parameter as follows:



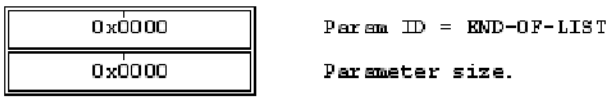
Bit 0 defines the scknowledged alarm state.
Bit 0 = 0 if the alarm has not been acknowledged
Bit 0 = 1 if the alarm has been acknowledged
All other bits are reserved.

The Point ID Hint parameter is:



The gateway sends this parameter if it cannot translate the alarm drop number into its short name form. This parameter indicates incomplete translation tables in the gateway.

The End of the List parameter is:



No more information about this alarm follows.

5.3 Process Control Command Requests

Process Control Commands are operator initiated actions such as pushbutton or setpoint commands. No more than ten (10) commands per second should be sent to any one turbine controller. This should be sufficient to handle operator initiated commands .

Two types of process control command request messages sent from the DCS:

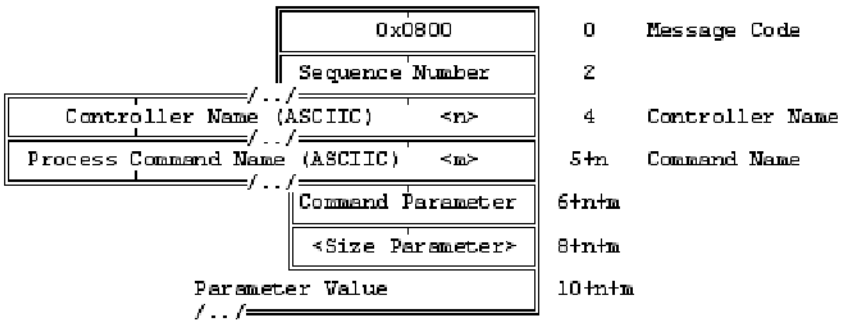
- pushbutton commands, such as Start/Stop and Raise/Lower
- setpoint target commands, such as Speed Target and Load Target

Both control command request messages can be blocked either by the gateway or the process controller. The DCS should monitor feedback signals to determine if the command request has been acted upon by the process controller.

To specify the number of scans for a pushbutton, use the setpoint command (the Mark V controller defaults to 4).

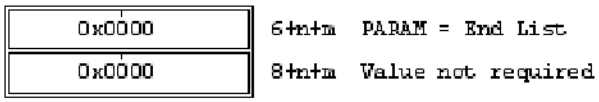
The gateway responds to a process control command request by a control request ACK/NAK response message.

The process control command request format is:

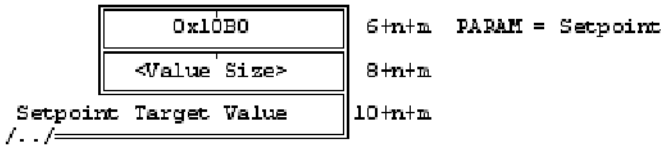


For Mark V systems, the controller names are typically T1, T2, ..Tn. The Process Command Name is the name of the variable to be changed.

The pushbutton command parameter is:



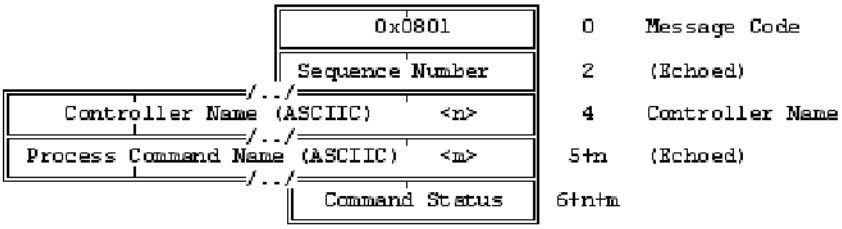
The setpoint target command parameter format is:



5.3.1 Process Control Command ACK/NAK Response

The gateway responds to a process control command request with a command request ACK/NAK response. This informs the DCS that the command is valid, and has been successfully sent. To confirm that the controller has the command, look for the associated alarm queue change message.

The process control command request ACK/NAK response format is:



The Sequence Number, Controller Name, and Process Command Name are echoed from the process control command request.

Process Control Command ACK/NAK Codes

Code	Description
0	Success. Requested command sent to the process controller
+1	Request received by the gateway, but communication with the process controller is not currently possible
-1	Unknown controller name
-2	Function not supported by the process controller
-3	Invalid process command name
-4	Invalid parameter, for example, a value for a pushbutton or no value for a setpoint target
-5	Internal gateway error
-6	Permission violation due to control hierarchy

Note No more than 10 individual process control command requests per second should be sent to any one controller.

6 Application Notes

This section contains two examples of programs to test the HMI GSM interface, plus application notes on networking and Telnet.

6.1 Sample Program – Alarms

This sample GSM program shows the process alarm messages, event messages, and SOE messages. A program of this type can also allow the user to send alarm commands to the controllers.

```
/*
       (C) COPYRIGHT 1998 GENERAL ELECTRIC COMPANY

       All rights reserved, including copyrights and trade secrets.
       No part of this program or information may be reproduced,
       transmitted, transcribed, stored in a retrieval system, or
       translated into any language, in any form or by any means,
       electronic, mechanical, magnetic, optical, chemical, manual
       or otherwise, or disclosed to others without the prior
       written permission of General Electric Company,
       1 River Road, Schenectady, NY 12345.
*/
/*    gsm_alm - This is a simple test program that interfaces to the exception
 *    stream for Process Alarms, Events, and SOEs.
 *    It will show the user the messages, and (for alarms) will allow the user
 *    to send alarm commands back to the units.
 */

#define WIN32_LEAN_AND_MEAN
#define NOSERVICE

#include <stdio.h>
#include <conio.h>
#include <string.h>
#include <process.h>
#include <winsock.h>
#pragma hdrstop

typedef struct
       {
       ULONG       seconds;
       ULONG       u_seconds;
       } GEDSTIME;
BOOL XuGedsTimeToFileTime(GEDSTIME *p1, FILETIME *p2);
BOOL XuFileTimeToAsciiFixed(FILETIME *p1, char *p2);

#define GSM_SOCKET 768

//#define VERBOSE

typedef unsigned short int UWORD;
typedef unsigned char UBYTE;
```

```
/*
*  G L O B A L     T Y P E S
*/

#pragma pack(push, 1)
typedef struct _ALARM_CCELL
      {
      UWORD       drop_number;
      UBYTE       command;
      } ALARM_CCELL;
typedef struct _ALARM_COMMAND_EX
      {
      UBYTE        unit;
      UBYTE        msg_type;
      UBYTE        num_records;
      ALARM_CCELL HiHicommand[12];
      } ALARM_COMMAND_EX;
#pragma pack(pop)

/*
*  G L O B A L     V A R I A B L E S
*/
char                dbuf                // Buffer for GSM msgs.
                    [4096];
char *buf = dbuf+2;
int                       ms-           // How many bytes we read
                          gc;
ALARM_               cmd;        // If we need to send a command
COMMAND_EX
UBY-                      cmd_msg_      // Command message type
TE                        type;
```

```
SOCKET gsm_socket;

char units[20][40];
unsigned short seq_num[20];
int seq_num_valid[20];
int ucount;
int waiting =1;
void socket_rcv(char *rcv_buf);
void socket_send(char *send_buf, int bytes);
int name_flag = 0,nocmd = 0;

//————————————————————————————————————————————————————————————————————————————

char rcv_buf[4096];

void ReceiveThread(void *unused) {
unsigned short gsm_code,code,size,*num,count,seqnum;
FILETIME time;
char msg[84],pname[40];
char time_string[40],uname[40],*c;
int drop,state,reason,ack,offset,lock,unit,status;
char *pos;
#ifdef VERBOSE
int links,utype;
#endif

time_string[0] = '\0';
while (1) {
       socket_rcv(rcv_buf);
       pos = rcv_buf;
       gsm_code = *(unsigned short *)pos;
       pos += 2;
       if (gsm_code == 0x701) { // alm cmd reply
              num = (unsigned short *)pos;
              pos += 2;
              count = *pos++;
              for (c=uname;count;count-) {
                     *c++ = *pos++;
              }
              *c = '\0';
              num = (unsigned short *)pos;
              pos += 2;
              num = (unsigned short *)pos;
              pos += 2;
              status = *(short *)pos;
              if (status != 0)
                     printf("\n*** Alarm Command failed, status: %d\n\n",status);
              pos += 2;
       }
       else if (gsm_code == 0x101) { // unit info
```

```
        ucount = 0;
        num = (unsigned short *)pos;
        pos += 2;
        pos += 3; // skip reserved and NULL unit
        pos += 4; // skip GSM_LIST and size
        while (1) {
               code = *(unsigned short *)pos;
               pos += 2;
               if (code == 0) break;
               size = *(unsigned short *)pos;
               pos += 2;
               while (1) {
                      code = *(unsigned short *)pos;
                      if (code == 0) {
                            pos += 4;
                            break;
                      }
                      if (code == 0x8100) break;
                      pos += 2;
                      size = *(unsigned short *)pos;
                      pos += 2;
                      switch(code) {
                      case 0x1000:
                            strncpy(units[ucount],pos,size);
                            units[ucount++][size] = '\0';
                            break;
                      case 0x1010:
                            brea-
                            k;
                      case 0x1020:
                            break;
                      }
                      pos += size;
               }
        }
        waiting = 0;
}
else if (gsm_code == 0x302 || gsm_code == 0x702) { // alarm msg/dump
        if (gsm_code == 0x302) {
               gsm_code == 2;
        }
        num = (unsigned short *)pos;
        pos += 2;
        count = *pos++;
        for (c=uname;count;count-) {
               *c++ = *pos++;
        }
        *c = '\0';
        for (unit=0;unit<20;unit++) {
               if (!stricmp(uname,units[unit])) break;
        }
        pos += 2; // skip reserved
        pos += 2; // skip GSM_LIST
        pos += 2; // skip list size;
printf("\nMessage Received: Unit=%s MsgType=%04hx\n",uname,gsm_code);
```

```
            while (1) {
                  code = *(unsigned short *)pos;
                  pos += 2;
                  if (code == 0) break;
                  size = *(unsigned short *)pos;
                  pos += 2;
                  *msg = '\0';
                  *pname = '\0';
                  while (1) {
                        code = *(unsigned short *)pos;
                        if (code == 0) {
                              pos += 4;
                              break;
                        }
                        if (code == 0x8300) break;
                        pos += 2;
                        size = *(unsigned short *)pos;
                        pos += 2;
                        switch(code) {
                        case 0x1030:
                              strncpy(pname,pos,size);
                              pname[size] = '\0';
                              break;
                        case 0x1040:
                              XuGedsTimeToFileTime((GEDSTIME *)pos,&time);
                              XuFileTimeToAsciiFixed(&time, time_string);
                              break;
                        case 0x1050:
                              drop = *(unsigned short *)pos;
                              break;
                        case 0x1060:
                              state = *(unsigned char *)pos;
                              break;
                        case 0x1070:
                              lock = *(unsigned char *)pos;
                              break;
                        case 0x1080:
                              reason = *(unsigned char *)pos;
                              break;
                        case 0x1090:
                              strncpy(msg,pos,size);
                              msg[size] = '\0';
                              break;
                        case 0x10c0:
                              ack = *(unsigned char *)pos;
                              break;
                        case 0x10a0:
                              seqnum = *(unsigned char *)pos;
                              if (gsm_code == 0x302 && seqnum !=
++seq_num[unit]) {
                                    puts("\n*** sequence number mismatch");
                                    printf("server: %hu local:
%hu\n\n",seqnum,seq_num[unit]);
                                    seq_num[unit] = seqnum;
```

```
                        }
                        break;
                  }
                  pos += size;
            }
      if (reason == 255)
            printf("—CLEAR ALARM QUEUE—\n");
      else {
            if (!name_flag)
                  printf(" %s Drop=%5d State=%d%c%c (Reason=%d)\n",
                        time_string,
                        drop,
                        state,
                        ack ? ' ' : '*',
                        lock ? 'L' : ' ',
                        reason);
            else
                  printf(" %s Name=%-12s State=%d%c%c (Reason=%d)\n",
                        time_string,
                        pname,
                        state,
                        ack ? ' ' : '*',
                        lock ? 'L' : ' ',
                        reason);
            if (*msg) puts(msg);
            }
      }
}
else if (gsm_code == 0x402 || gsm_code == 0x502) { // event/soe msg
      num = (unsigned short *)pos;
      pos += 2;
      count = *pos++;
      for (c=uname;count;count-) {
            *c++ = *pos++;
      }
      *c = '\0';
      pos += 2; // skip reserved
      pos += 2; // skip GSM_LIST
      pos += 2; // skip list size;
printf("\nMessage Received: Unit=%s MsgType=%04hx\n",uname,gsm_code);
      while (1) {
            code = *(unsigned short *)pos;
            pos += 2;
            if (code == 0) break;
            size = *(unsigned short *)pos;
            pos += 2;
            *msg = '\0';
            *pname = '\0';
            while (1) {
                  code = *(unsigned short *)pos;
                  if (code == 0) {
                        pos += 4;
                        break;
```

```
                        }
                        if (code == 0x8400 || code == 0x8500) break;
                        pos += 2;
                        size = *(unsigned short *)pos;
                        pos += 2;
                        switch(code) {
                        case 0x1030:
                              strncpy(pname,pos,size);
                              pname[size] = '\0';
                              break;
                        case 0x1040:
                              XuGedsTimeToFileTime((GEDSTIME *)pos,&time);
                              XuFileTimeToAsciiFixed(&time, time_string);
                              break;
                        case 0x1060:
                              state = *(unsigned char *)pos;
                              break;
                        case 0x10d0:
                              offset = *(unsigned short *)pos;
                              break;
                        case 0x1090:
                              strncpy(msg,pos,size);
                              msg[size] = '\0';
                              break;
                        }
                        pos += size;
                  }
            if (*pname)
                  printf(" %s Name=%-12s State=0x%.2X\n",
                        time_string,
                        pname,
                        state);
            else
                  printf(" %s Offset=0x%.4X State=0x%.2X\n",
                        time_string,
                        offset,
                        state);
            if (*msg) puts(msg);
            }
      }
}
}
void heartbeat(void *unused) {
      char buf[256],*pos;
      int size;

      pos = buf+2;
      *(unsigned short *)pos = 0x0200; // heartbeat
      pos += 2;
      *(unsigned short *)pos = 0x0000; // seq num
      pos += 2;      // unit
                     name
      *pos++ = 0;
      size = pos-buf;
      while (1) {
```

```
        Sleep(20000);
        socket_send(buf,size);
    }
```

```
}
/-
/————————————————————————————————————————————————————————————————————————-
———
void do_help(void)
{
printf("\n");
printf("GSM_ALM - Alarm Dump of exception messages.\n");
printf("\n");
printf(" This program will display the exception messages received from each unit\n");
printf(" as they arrive. By default it will show the Process Alarm exceptions,\n");
printf(" but this can be changed by supplying a command line parameter indicating\n");
printf(" the exception list to display.\n");
printf("\n");
printf(" Commands can be sent back to the unit to silence, acknowledge, or reset\n");
printf(" alarms. Enter \"?\" while running for a list of valid commands.\n");
printf("\n");
printf(" COMMAND FORMAT: GSM_ALM host [list]\n");
printf("\n");
printf("    host - dotted decimal IP address or hostname of GSM server\n");
printf("    [list] indicates the exception list to display, which can be:\n");
printf("        PALARM or PROCESS......Displays the Process Alarm exceptions\n");
printf("        EVENT..................Displays the Event List exceptions \n");
printf("        SOE....................Displays the Sequence of Events exceptions\n");
printf("        NAME...................Displays the name instead of drop number\n");
printf("        MSG....................Displays the alarm message\n");
printf("\n");
}    /* End of do_help */
//————————————————————————————————————————————————————————————————————————
void do_key_help(void)
{
printf("\n");
printf(" Available commands:\n");
printf("    A - Send an ACK command to the current unit\n");
printf("    N - Send <n>; ACK commands to the current unit\n");
printf("    R - Send a RESET command to the current unit\n");
printf("    L - Send a LOCK command to the current unit\n");
printf("    U - Send an UNLOCK command to the current unit\n");
printf("    D - Send a DUMP REQUEST to the current unit\n");
printf("    # - Change the current unit number\n");
printf("    1 - Send an ACK ALL to the current unit\n");
printf("    2 - Send a RESET ALL to the current unit\n");
printf("    3 - Send a SILENCE to the current unit\n");
printf("    8 - Send 64 ACK SINGLE to the current unit\n");
printf("    9 - Send 64 RESET SINGLE to the current unit\n");
printf("    ? - Help, show this message\n");
printf("        A blank line is ignored, no action is taken.\n");
printf("        <ESC>; will exit the program, as will CTRL+C or CTRL+Z\n");
printf("\n");
}    /* End of do_key_help */


/*————————————————————————————————————————————————————————————
 * Function: GetAddr()
 *
 * Description: Given a string, it will return an IP address.
 *     - first it tries to convert the string directly
 *     - if that fails, it tries o resolve it as a hostname
 *
```

```
        /* check for a dotted-IP address string */
        lAddr = inet_addr (szHost);
        /* If not an address, then try to resolve it as a hostname */
        if ((lAddr == INADDR_NONE) &&
                (strcmp (szHost, "255.255.255.255")))
        {
                lpstHost = gethostbyname(szHost);
                if (lpstHost) { /* success */
                        lAddr = *((u_long FAR *) (lpstHost->;h_addr));
                }
                else
                {
                        lAddr = INADDR_ANY; /* failure */
                }
        }
 }
 return (lAddr);
} /* end GetAddr() */
void errmsg(char *msg) {
        puts(msg);
}

static int send_bytes(SOCKET,char *,int);
void socket_send(char *send_buf, int bytes) {
        unsigned short size;

        /* precede the message with its size */
        size = bytes-2;
        *(unsigned short *)(send_buf) = size;
        //if (send_bytes(gsm_socket,(char *)&size,2) == SOCKET_ERROR) {
        //      dperror("send");
        //      if (closesocket(gsm_socket) == SOCKET_ERROR) {
        //              dperror("close");
        //      }
        //      exit(0);
        //}
        if (send_bytes(gsm_socket,send_buf,bytes) == SOCKET_ERROR) {
                dperror("send");
                if (closesocket(gsm_socket) == SOCKET_ERROR) {
                        dperror("close");
                }
                exit(0);
        }
}
static int send_bytes(SOCKET s,char *buf,int bytes) {
        int count;

        while (bytes >; 0) {
                count = send(s,buf,bytes,0);
                if (count == SOCKET_ERROR) {
                        return SOCKET_ERROR;
                }
                else {
```

```
                        bytes -= count;
                        buf  += count;
                }
        }
        return count;
}

int read_bytes(SOCKET s,char *buf,int count);

void socket_rcv(char *rcv_buf) {
        int bytes;

        if (read_bytes(gsm_socket,rcv_buf,2) == SOCKET_ERROR) {
                exit(1);
        }
        bytes = *(unsigned short *)rcv_buf;
        if (read_bytes(gsm_socket,rcv_buf,bytes) == SOCKET_ERROR) {
                exit(1);
        }
}

static int read_bytes(SOCKET s,char *buf,int count) {
        int bytes;

        while (count >; 0) {
                bytes = recv(s,buf,count,0);
                if (bytes == 0) {
                        errmsg("remote disconnect");
                        if (closesocket(s) == SOCKET_ERROR) {
                                dperror("close");
                        }
                        return SOCKET_ERROR;
                }
                if (bytes == SOCKET_ERROR) {
                        dperror("recv");
                        if (closesocket(s) == SOCKET_ERROR) {
                                dperror("close");
                        }
                        return SOCKET_ERROR;
                }
                count -= bytes;
                buf += bytes
        }
        return 1
}

//---------------------------------------------------------------------------
int main(int argc, char *argv[])
{
int                     i;
int                     item;
int                     drop;
int                     default_
                        unit;
```

```
WORD wVersionRequested;
WSADATA wsaData;
SOCKADDR_IN sin;
int err;
unsigned short alport = GSM_SOCKET;
unsigned short gsm_request_code = 0x300;
unsigned short seq_num=0;
char *pos;
int size;
int msg_flag = 0;

wVersionRequested = MAKEWORD(1, 1);
err = WSAStartup(wVersionRequested, &wsaData);

if (err != 0)
        {
       /* Tell the user that we couldn't find a useable */
       /* winsock.dll. */
       printf("I can't find winsock.dll.\n");
       return(-1);
       }
/* Confirm that the Windows Sockets DLL supports 1.1.*/
/* Note that if the DLL supports versions greater */
/* than 1.1 in addition to 1.1, it will still return */
/* 1.1 in wVersion since that is the version we */
/* requested. */
if ( LOBYTE( wsaData.wVersion ) != 1 ||
       HIBYTE( wsaData.wVersion ) != 1 ) {
       /* Tell the user that we couldn't find a useable */
       /* winsock.dll. */
       WSACleanup();
        printf("Bad winsock.dll version.\n");
       return(-2);
}
/* The Windows Sockets DLL is acceptable. Proceed. */

// The Windows Sockets bind function associates a local address with a socket.


// Prepare
sin.sin_family = AF_INET;
sin.sin_addr.s_addr = GetAddr("127.0.0.1");
sin.sin_port = htons(alport);     // 13=daytime 7=echo 768=GSM
// The Windows Sockets socket function creates a socket.
gsm_socket = socket(PF_INET, SOCK_STREAM, 0);
if (gsm_socket == INVALID_SOCKET) {
       dperror("socket");
       return(-3);
```

```
}

//
// Check to see what program we wish to be. Default to PROCESS ALARM
//

cmd_msg_type = 2;

for (i = 1; i < argc; i++)
       {
       if (!strcmp(argv[i], "/?"))
              {
              do_help();
              exit(0);
              }
       if ((!stricmp(argv[i], "PALARM"))
       || (!stricmp(argv[i], "PROCESS")))
              {
              cmd_msg_type = 2;
              gsm_request_code = 0x300;
              nocmd = 0;
              continue;
              }
       if ((!stricmp(argv[i], "DALARM"))
       || (!stricmp(argv[i], "DIAG")))
              {
              puts("Diagnostic alarms are not available via GSM");
              continue;
              }
       if (!stricmp(argv[i], "HOLD"))
              {
              puts("Hold Lists are not available via GSM");
              continue;
              }
       if (!stricmp(argv[i], "EVENT"))
              {
              cmd_msg_type = 8;
              nocmd = 1;
              gsm_request_code = 0x500;
              continue;
              }
       if (!stricmp(argv[i], "SOE"))
              {
              cmd_msg_type = 10;
              nocmd = 1;
              gsm_request_code = 0x400;
              continue;
              }
       if (!stricmp(argv[i], "MSG"))
              {
              msg_flag ^= 1;
              continue;
              }
       if (!stricmp(argv[i], "NAME"))
```

```
                {
                name_flag ^= 1;
                continue;
                }
        if ((sin.sin_addr.s_addr = GetAddr(argv[i])) == INADDR_ANY)
                {
                sin.sin_addr.s_addr = GetAddr("127.0.0.1");
                printf("...Unknown command line parameter \"%s\" ignored.\n",
                    argv[i]);
                printf("...Hint: Try ( PALARM | DALARM | DIAG | EVENT | SOE )\n");
                printf("...Hint: Run as \"ALMDUMP1 /?\" for help.\n");
                }
        }
if (connect(gsm_socket,(const struct sockaddr *)&sin, sizeof(struct sockaddr))
== SOCKET_ERROR)
        {
        dperror("connect");
        return(-1);
        }
// get unit list
pos = buf+2;
*(unsigned short *)pos = 0x0100; // supported units
pos += 2;
*(unsigned short *)pos = 0x0001; // seq num
pos += 2;  // unit name
*pos++ = 0;
size = pos-buf;
socket_send(buf,size);
//
//
// Launch the thread the displays messages received. We take a clue
// from whether there are commands available to determine which version
// (ALARM | EVENT) to launch.
//
_beginthread(ReceiveThread,0,NULL);
_beginthread(heartbeat,0,NULL);

// wait for supported units response
while (waiting) Sleep(500);
puts("Available Units:");
for (i=0;i<ucount;i++) {
        //units[i][0]++;
        puts(units[i]);
        seq_num_valid[i] = 0;
}

default_unit = 0;
for (i=0;units[i][0];i++) {
```

```
        pos = buf+2;
        *(unsigned short *)pos = gsm_request_code; // gsm alarm data
        pos += 2;
        *(unsigned short *)pos = i+1; // seq num
        pos += 2;  // unit name
        *pos++ = strlen(units[i]);
        strcpy(pos,units[i]);
        pos += strlen(units[i]);
        *(unsigned short *)pos = 0x0000; // establish func
        pos += 2;
        *(unsigned short *)pos = msg_flag; // options
        pos += 2;
        size = pos-buf;
        socket_send(buf,size);
}
// Now start the super simple loop if we are watching events.
if (nocmd)                                              // No commands, must be event
type
        {
        while (1)
                {
                printf("Hit <ESC; to exit: ");
                item = getch();
                printf("\n");
                switch(item)
                        {
                        case 0x0D:                                      // Blank line is a
NOP, don't complain
                                break;
                        case 'C' - 0x40:                        // Control-C is an exit
                        case 0x1B:                                        // ...so is <ESCAPE>
                        case EOF:                                       // ...so is Control-Z
                                exit(0);
                        case '?':
                                do_key_help();
                                printf("\nNote: No commands are supported for EVENTs or
                                SOEs.\n\n");
                                break;
                        }
                }
        }
// If I fell through the above check I must be running in an ALARM mode.
cmd.unit = 0xff;
while (1)
        {
        Sleep(1000);
        printf("Unit %s Cmd (A=Ack, R=Reset, L=Lock, U=Unlock, D=Dump, #=Unit,
?=Help):",
                units[default_unit]);
        item = getch();
        printf("\n");
        switch(item)
                {
                case 0x0D:                                      // Blank line is a NOP,
don't complain
```

```
                    cmd.unit = 0xff;                            // ...but don't do
anything either
                    break;
              case 'C' - 0x40:                                  // Control-C is an exit
              case 0x1B:                                          // ...so is <ESCAPE>
              case EOF:                                          // ...so is Control-Z
                    exit(0);
              case '?':
                    {
                    do_key_help();
                    cmd.unit = 0xff;
                    break;
                    }
              case '1':                 // Undocumented ACK ALL
                    {
                    printf("Sending ACK ALL to unit #%d\n", default_unit);
                    cmd.unit = default_unit;
                    cmd.msg_type = cmd_msg_type;
                    cmd.num_records = 1;
                    cmd.command[0].drop_number = 0xffff;
                    cmd.command[0].command = 4;
                    break;
                    }
              case '2':                 // Undocumented RESET ALL
                    {
                    printf("Sending RESET ALL to unit #%s\n",
units[default_unit]);
                    cmd.unit = default_unit;
                    cmd.msg_type = cmd_msg_type;
                    cmd.num_records = 1;
                    cmd.command[0].drop_number = 0;
                    cmd.command[0].command = 6;
                    break;
                    }
              case '3':                 // Undocumented SILENCE
                    {
                    printf("Sending SILENCE to unit #%s\n", units[default_unit]);
                    cmd.unit = default_unit;
                    cmd.msg_type = cmd_msg_type;
                    cmd.num_records = 1;
                    cmd.command[0].drop_number = 0;
                    cmd.command[0].command = 10;
                    cmd.unit = 0;
                    break;
                    }
              case '#':
```

```
        {
        printf("UNIT ->; Enter unit number: ");
        scanf("%d", &default_unit);
        if (default_unit >;= ucount) {
             default_unit = 0;
        }
        cmd.unit = 0xff;
        break;
        }
case 'a':
case 'A':
        {
        printf("ACK ->; Enter drop number: ");
        scanf("%d", &drop);
        cmd.unit = default_unit;
        cmd.msg_type = cmd_msg_type;
        cmd.num_records = 1;
        cmd.command[0].drop_number = drop;
        cmd.command[0].command = 7;
        break;
        }
case 'n':
case 'N':
        {
        printf("ACK ->; Enter number of alarms to ACK: ");
        scanf("%d", &drop);
        cmd.unit = default_unit;
        cmd.msg_type = cmd_msg_type;
        cmd.num_records = 1;
        cmd.command[0].drop_number = drop;
        cmd.command[0].command = 4;
        break;
        }
case 'r':
case 'R':
        {
        printf("RESET ->; Enter drop number: ");
        scanf("%d", &drop);
        cmd.unit = default_unit;
        cmd.msg_type = cmd_msg_type;
        cmd.num_records = 1;
        cmd.command[0].drop_number = drop;
        cmd.command[0].command = 8;
        break;
        }
case 'l':
case 'L':
```

```
                        {
                        printf("LOCK ->; Enter drop number: ");
                        scanf("%d", &drop);
                        cmd.unit = default_unit;
                        cmd.msg_type = cmd_msg_type;
                        cmd.num_records = 1;
                        cmd.command[0].drop_number = drop;
                        cmd.command[0].command = 2;
                        break;
                        }
                case 'u':
                case 'U':
                        {
                        printf("UNLOCK ->; Enter drop number: ");
                        scanf("%d", &drop);
                        cmd.unit = default_unit;
                        cmd.msg_type = cmd_msg_type;
                        cmd.num_records = 1;
                        cmd.command[0].drop_number = drop;
                        cmd.command[0].command = 3;
                        break;
                        }
                case 'd':
                case 'D':
                        {
                        printf("Sending DUMP REQUEST to unit #%s\n",
units[default_unit]);
                        cmd.unit = default_unit;
                        cmd.msg_type = cmd_msg_type;
                        cmd.num_records = 1;
                        cmd.command[0].drop_number = msg_flag;
                        cmd.command[0].command = 255;
                        break;
                        }
                default:
                        printf("\n Unknown command - ignored.\n");
                        cmd.unit = 0xff;
                }
        if (cmd.unit != 0xff)
                {
                pos = buf+2;
                *(unsigned short *)pos = 0x700;
                pos += 2;
                *(unsigned short *)pos = seq_num++; // seq num
                pos += 2;         // unit name
                *pos++ = strlen(units[default_unit]);
                strcpy(pos,units[default_unit]);
                pos += strlen(units[default_unit]);
                *(unsigned short *)pos = cmd.command[0].command; // establish func
                pos += 2;
                *(unsigned short *)pos = cmd.command[0].drop_number; // options
                pos += 2;
                size = pos-buf;
                socket_send(buf,size);
                }
```

```
        }
}

BOOL XuFileTimeToAsciiFixed(FILETIME *p1, char *p2)
{
SYSTEMTIME        s;

*p2 = 0;
if (!FileTimeToSystemTime(p1, &s))
        return(0);
sprintf(p2, "%.2d-mmm-%.4d %.2d:%.2d:%.2d.%.3d",
        s.wDay, s.wYear, s.wHour, s.wMinute, s.wSecond, s.wMilliseconds);
switch(s.wMonth)
        {
        case 1: memcpy(p2+3, "JAN", 3);                          break;
        case 2: memcpy(p2+3, "FEB", 3);                          break;
        case 3: memcpy(p2+3, "MAR", 3);                          break;
        case 4: memcpy(p2+3, "APR", 3);                          break;
        case 5: memcpy(p2+3, "MAY", 3);                          break;
        case 6: memcpy(p2+3, "JUN", 3);                          break;
        case 7: memcpy(p2+3, "JUL", 3);                          break;
        case 8: memcpy(p2+3, "AUG", 3);                          break;
        case 9: memcpy(p2+3, "SEP", 3);                          break;
        case 10: memcpy(p2+3, "OCT", 3);                         break;
        case 11: memcpy(p2+3, "NOV", 3);                         break;
        case 12: memcpy(p2+3, "DEC", 3);                         break;
        default: memcpy(p2+3, "???", 3);
        }
return(1);
} /* End of XuFileTimeToAsciiFixed */
//-------------------------------------------------------------------------
BOOL XuGedsTimeToFileTime(GEDSTIME *p1, FILETIME *p2)
{
if (p1->u_seconds > 999999)
        {
        SetLastError(ERROR_INVALID_PARAMETER);
        return(0);
        }
*(DWORDLONG *)p2 =
        ((((DWORDLONG)p1->seconds * 1000000) + p1->u_seconds) * 10
        + (DWORDLONG) 0x019DB1DED53E8000);                       /* Clunks at 1/1/1970 */
return(1);
}     /* End of XuGedsTimeToFileTime */
     /* End of AlarmDump1 */
//-----------------------------------------------------------
```

6.2 Sample Program – View Data Lists

This sample GSM program can be used to display selected data points.

```
/*
     (C) COPYRIGHT 1998 GENERAL ELECTRIC COMPANY

     All rights reserved, including copyrights and trade secrets.
     No part of this program or information may be reproduced,
     transmitted, transcribed, stored in a retrieval system, or
     translated into any language, in any form or by any means,
     electronic, mechanical, magnetic, optical, chemical, manual
     or otherwise, or disclosed to others without the prior
     written permission of General Electric Company,
     1 River Road, Schenectady, NY 12345.
*/

/* gsm_view - This is a simple test program that demonstrates the GSM
* protocol for periodic data lists.
*/
#define WIN32_LEAN_AND_MEAN
#define NOSERVICE
#include <windows.h>
#include <stdlib.h>
#include <stdio.h>
#include <string.h>
#include <process.h>
#include <winsock.h>

#pragma hdrstop

#define GSM_SOCKET 768
#define BUF_SIZE 4096
//#define VERBOSE

typedef char STRING80[80];

typedef struct
     {
     ULONG          seconds;
     ULONG          u_seconds;
     } GEDSTIME;
```

```
BOOL XuGedsTimeToFileTime(GEDSTIME *p1, FILETIME *p2);
BOOL XuFileTimeToAsciiFixed(FILETIME *p1, char *p2);

SOCKET gsm_socket;
char buf[BUF_SIZE];
/*-------------------------------------------------------------------
 * Function: GetAddr()
 *
 * Description: Given a string, it will return an IP address.
 *     - first it tries to convert the string directly
 *     - if that fails, it tries o resolve it as a hostname
 *
 * WARNING: gethostbyname() is a blocking function
 */
u_long GetAddr (LPSTR szHost) {
     LPHOSTENT lpstHost;
     u_long lAddr = INADDR_ANY;

     /* check that we have a string */
     if (*szHost)
            /* check for a dotted-IP address string */
            lAddr = inet_addr (szHost);

            /* If not an address, then try to resolve it as a hostname */
            if ((lAddr == INADDR_NONE) &&
                   (strcmp (szHost, "255.255.255.255")))
            {
                   lpstHost = gethostbyname(szHost);
                   if (lpstHost) { /* success */
                          lAddr = *((u_long FAR *) (lpstHost->;h_addr));
                   }
                   else
                   {
                          lAddr = INADDR_ANY; /* failure */
                   }
            }
     }
     return (lAddr);
} /* end GetAddr() */

/*-------------------------------------------------------------------
 * Function: send_bytes()
 * Description: Sends data out the given socket.
 * Returns SOCKET_ERROR if an error occurs.
*/
static int send_bytes(SOCKET s,char *buf,int bytes) {
     int count;

     while (bytes >; 0) {
```

```
            count = send(s,buf,bytes,0);
            if (count == SOCKET_ERROR) {
                  return SOCKET_ERROR;
            }
            else {
                  bytes -= count;
                  buf  += count;
            }
      }
      return count;
}
/*-------------------------------------------------------------------
 * Function: socket_send()
 * Description: Sends a GSM message to the GSM server.
*/
void socket_send(char *send_buf, int bytes) {
      unsigned short size;


      /* precede the message with its size */
      size = bytes-2;
      *(unsigned short *)(send_buf) = size;

      /* send the message size */
//    if (send_bytes(gsm_socket,(char *)&size,2) == SOCKET_ERROR) {
//          dperror("send");
//          if (closesocket(gsm_socket) == SOCKET_ERROR) {
//                dperror("close");
//          }
//          exit(0);
//    }
      /* send the message */
      if (send_bytes(gsm_socket,send_buf,bytes) == SOCKET_ERROR) {
            dperror("send");
            if (closesocket(gsm_socket) == SOCKET_ERROR) {
                  dperror("close");
            }
            exit(0);
      }
}
/*-------------------------------------------------------------------
 * Function: read_bytes()
 * Description: Reads data from the given socket.
 * Returns SOCKET_ERROR if an error occurs.
*/
static int read_bytes(SOCKET s,char *buf,int count) {
```

```
	int bytes;

	while (count > 0) {
		bytes = recv(s,buf,count,0);
		if (bytes == 0) {
			errmsg("remote disconnect");
			if (closesocket(s) == SOCKET_ERROR) {
				dperror("close");
			}
			return SOCKET_ERROR;
		}
		if (bytes == SOCKET_ERROR) {
			dperror("recv");
			if (closesocket(s) == SOCKET_ERROR) {
				dperror("close");
			}
			return SOCKET_ERROR;
		}
		count -= bytes;
		buf += bytes;
	}
	return 1;
}
/*---------------------------------------------------------------------
 * Function: socket_rcv()
 * Description: Receives a GSM message from the server
*/
void socket_rcv(char *rcv_buf) {
	int bytes;

	/* read the message size */
	if (read_bytes(gsm_socket,rcv_buf,2) == SOCKET_ERROR) {
		exit(1);
	}
	bytes = *(unsigned short *)rcv_buf;

	/* read the message */
	if (read_bytes(gsm_socket,rcv_buf,bytes) == SOCKET_ERROR) {
		exit(1);
	}
}

/*---------------------------------------------------------------------
 * Function: heartbeat()
 * Description: Sends a heartbeat to the GSM server
 * every 20 seconds
*/
void heartbeat(void *unused) {
```

```
      char buf[256],*pos;
      int size;

      pos = buf+2;
      *(unsigned short *)pos = 0x0200; // heartbeat
      pos += 2;
      *(unsigned short *)pos = 0x0000; // seq num
      pos += 2;         // unit name
      *pos++ = 0;
      size = pos-buf;
      while (1) {
            Sleep(20000);
            socket_send(buf,size);
      }
}
//---------------------------------------------------------------------------
void main (int argc, char *argv[])
{
int             i,cnt;
STRING80        host,signal;
char *point,
*unit;
STRING80        time_string;
FILETIME        time;
WORD wVersionRequested;
WSADATA wsaData;
int err,size;
SOCKADDR_IN sin;
unsigned short alport = GSM_SOCKET;
char uname[40];
char *pos;
unsigned short *num,code,list_name,sval;
unsigned long lval;
double fval;
int links,utype;

/* load WinSock */
wVersionRequested = MAKEWORD(1, 1);
err = WSAStartup(wVersionRequested, );

if (err != 0) {
      printf("I can't find winsock.dll.\n");
      exit(1);
}

if ( LOBYTE( wsaData.wVersion ) != 1 || HIBYTE( wsaData.wVersion ) != 1 ) {
      WSACleanup();
      printf("Bad winsock.dll version.\n");
      exit(1);
```

```
}

/* look at the command line */
for (i = 1; i < argc; i++) {
     strupr(argv[i]);
     if ((argv[i][0] != '/') && (argv[i][0] != '-')) {
            strcpy(host, argv[i]);
            continue;
     }
}

/* ask user for signal name */
if (1) {
     printf("Enter unit:pointname: ");
     gets(signal);
     for (i = 0; signal[i]; i++) {
            if (signal[i] < ' ')
                  signal[i] = '\0';
     }
}

/* extract unit name */
unit = strtok(signal,":");
point = strtok(NULL,"");

// Prepare socket
sin.sin_family = AF_INET;
if (argc ==2)
     sin.sin_addr.s_addr = GetAddr(host);
else
     sin.sin_addr.s_addr = GetAddr("127.0.0.1");
sin.sin_port = htons(alport);

gsm_socket = socket(PF_INET, SOCK_STREAM, 0);
if (gsm_socket == INVALID_SOCKET) {
     dperror("socket");
     exit(1);
}

/* connect to the GSM server */
err = connect(gsm_socket, (const struct sockaddr *)&sin, sizeof(struct
sockaddr));
if (err) {
     dperror("connect");
     exit(1);
```

```
}

/* start the heartbeat */
_beginthread(heartbeat,0,NULL);

/* send supported units request */
pos = buf+2;
*(unsigned short *)pos = 0x0100; // supported units
pos += 2;
*(unsigned short *)pos = 0x0000; // seq num
pos += 2;          // unit name
*pos++ = 0;
socket_send(buf,pos-buf);

/* send periodic data request */
pos = buf+2;
*(unsigned short *)pos = 0x0600; // gsm periodic data
pos += 2;
*(unsigned short *)pos = 0x0000; // seq num
pos += 2;
strcpy(pos+1,unit); // unit name
*pos = strlen(unit);
pos += *pos + 1;
*(unsigned short *)pos = 0x0000; // establish func
pos += 2;
*(unsigned short *)pos = 1; // list name
pos += 2;
*(unsigned short *)pos = 3; // period code (in sec.)
pos += 2;
*(unsigned short *)pos = 0x1030; // gsm point name
pos += 2;
*(unsigned short *)pos = strlen(point);
pos += 2;
strcpy(pos,point);
pos += strlen(point);
*(unsigned long *)pos = 0;
pos +=4;
socket_send(buf,pos-buf);

/* start reading data from the server */

while (1) {
     /* read the GSM message */
     socket_rcv(buf);
     pos = buf;
     num = (unsigned short *)pos;
     pos += 2;
     if (*num == 0x101) {
            num = (unsigned short *)pos;
            pos += 2;
            pos += 3; // skip reserved and NULL unit
            pos += 4; // skip GSM_LIST and size
            while (1) {
```

```
                code = *(unsigned short *)pos;
                pos += 2;
                if (code == 0) break;
                size = *(unsigned short *)pos;
                pos += 2;
                while (1) {
                        code = *(unsigned short *)pos;
                        if (code == 0) {
                                pos += 4;
                                break;
                        }
                        if (code == 0x8100) break;
                        pos += 2;
                        size = *(unsigned short *)pos;
                        pos += 2;
                        switch(code) {
                        case 0x1000:
                                strncpy(uname,pos,size);
                                uname[size] = '\0';
                                break;
                        case 0x1010:
                                links = *(unsigned short *)pos;
                                break;
                        case 0x1020:
                                utype = *(unsigned char *)pos;
                                break;
                        }
                        pos += size;
                }
                printf("%10s links: %4d utype: %d\n",
                        uname,links,utype);
        }
}
else if (*num == 0x602) {
        num = (unsigned short *)pos;
        pos += 2;
        cnt = *pos++;
        while (cnt-) {
                pos++;
        }
        list_name = *(unsigned short *)pos;
        pos += 2; // skip list name
        pos += 4; // skip time ID
        XuGedsTimeToFileTime((GEDSTIME *)pos,&time);
        XuFileTimeToAsciiFixed(&time, time_string);
        pos += 8;
        i=0;
        while (1) {
```

```
                    code = *(unsigned short *)pos;
                    pos += 2;
                    if (code == 0) break;
                    size = *(unsigned short *)pos;
                    pos += 2;
                    if (size == 1) {
                            sval = *pos;
                            pos += 1;
                            printf("%s (%hu): %hu\n",point,size,sval);
                    }
                    if (size == 2) {
                            sval = *(unsigned short *)pos;
                            pos += 2;
                            printf("%s (%hu): %hu\n",point,size,sval);
                    }
                    else if (size == 4) {
                            //lval = *(unsigned long *)pos;
                            //pos += 4;
                            //printf("%s (%hu): %u\n",point,size,lval);
                            fval = *(float *)pos;
                            pos += 4;
                            printf("%s (%hu): %g\n",point,size,fval);
                    }
                    else if (size == 8) {
                            fval = *(double *)pos;
                            pos += 8;
                            printf("%s (%hu): %g\n",point,size,fval);
                    }
                    i++;
            }
     }
     else {
     }
}
closesocket(gsm_socket);

exit(0);
}  /* End of main program */

BOOL XuFileTimeToAsciiFixed(FILETIME *p1, char *p2)
{
SYSTEMTIME  s;
*p2 = 0;
if (!FileTimeToSystemTime(p1, &s))
     return(0);
sprintf(p2, "%.2d-mmm-%.4d %.2d:%.2d:%.2d.%.3d ",
     s.wDay, s.wYear, s.wHour, s.wMinute, s.wSecond, s.wMilliseconds);
switch(s.wMonth)
```

```
      {
      case 1: memcpy(p2+3, "JAN", 3);      break;
      case 2: memcpy(p2+3, "FEB", 3);      break;
      case 3: memcpy(p2+3, "MAR", 3);      break;
      case 4: memcpy(p2+3, "APR", 3);      break;
      case 5: memcpy(p2+3, "MAY", 3);      break;
      case 6: memcpy(p2+3, "JUN", 3);      break;
      case 7: memcpy(p2+3, "JUL", 3);      break;
      case 8: memcpy(p2+3, "AUG", 3);      break;
      case 9: memcpy(p2+3, "SEP", 3);      break;
      case 10: memcpy(p2+3, "OCT", 3);      break;
      case 11: memcpy(p2+3, "NOV", 3);      break;
      case 12: memcpy(p2+3, "DEC", 3);      break;
      default: memcpy(p2+3, "???", 3);
      }
return(1);
}  /* End of XuFileTimeToAsciiFixed */

//————————————————————————————————————————————————————————————————————————————
BOOL XuGedsTimeToFileTime(GEDSTIME *p1, FILETIME *p2)
{
if (p1->u_seconds > 999999)
      {
      SetLastError(ERROR_INVALID_PARAMETER);
      return(0);
      }
*(DWORDLONG *)p2 =
      ((((DWORDLONG)p1->seconds * 1000000) + p1->u_seconds) * 10
      + (DWORDLONG) 0x019DB1DED53E8000);
return(1);
}    /* End of XuGedsTimeToFileTime */
    /* End of AlarmDump1 */
//————————————————————————————————————————————————————————————————————————————
```

6.3 Networking

The GSM server uses TCP port 768 on all configured network interfaces on the gateway. A different TCP port number can be specified by including an option in the Options section of the CONFIG.DAT file in the Site Directory.

```
Options
 GSM_PORT = 769
```

Each GSM message must be preceded by a 2-byte integer indicating the size of the message. The recipient can use this size as an indication of how many bytes to read from the network. The maximum GSM message size that the gateway can send or receive is 4096 bytes.

6.3.1 TCP communications

There is no confusion over where messages are directed.

Conceptually, a TCP socket represents the endpoint of a virtual circuit. Each client socket has a corresponding socket on the server, and each client-server socket pair are the endpoints of a distinct virtual circuit. Thus each client has a separate communication path to the HMI. Responses to requests from Client A are sent only to Client A.

That the server is using only one port can be confusing. The fundamental concept of the TCP protocol is the virtual circuit, and that each virtual circuit is uniquely identified by a pair of endpoints. An endpoint is identified by a pair of integers (host and port) where host is the IP address of the host, and port is a TCP port number. Thus the same endpoint on the server can be used by multiple virtual circuits because each virtual circuit is identified by a pair of endpoints.

6.4 Telnet Interface

Telnet provides a GSM debugging tool.

The GSM server provides a simple Telnet interface that can be used to examine data in the server and trace messages passed to and from a client. To use the telnet interface, run a Telnet client and connect to the gateway on the GSM port (768 by default). When the connection is made, the server needs to be informed that the connection is from a Telnet client and not a GSM client. To do this, press the z key twice and the server responds with a command prompt. If the first two characters typed are not zz, the server assumes that a GSM client has connected and you will have to disconnect and try again.

Telnet Command Summary

Commands	Description
show client	Displays a list of all connected clients, including which controllers they have signed up for alarms, events, or periodic data
show log	Displays the contents of the file GSM log file
show list <client number> <controller name> <list number>	Displays the given periodic data list, including point name, point type, last reported value as well as the time the list was last sent to the client
trace <client number>	Toggle tracing of GSM messages sent to and received from the given client
trace next	Enables tracing of GSM messages sent to and received from the next client to connect to the server. This is a very useful command because it provides time to get ready for the data.
trace all	Enables tracing of GSM messages sent to and received from all clients that connect to the server. This acts as a toggle.
trace off	Turns all tracing off for this Telnet client
help	Provides a list of the GSM console commands
exit	Disconnect from the GSM server

The command line can be edited while it is being entered using the BACKSPACE (or CTRL+H) key. This works best if local echo has been enabled at the Telnet client.

The previous command can be repeated by entering a command line consisting of a single period (.) followed by ENTER. There is no way to edit the previous command line - only a way to repeat it.

The server responds to any input that it cannot understand by displaying a command summary (help) message.

6.5 Point ID Hint Parameter

The Point ID Hint parameter can be used as a point name in a periodic data request. The Point ID Hint parameter represents the Control Signal Database (CSDB) offset for a data point in a Mark V controller. When this offset is converted to ASCII in base ten, it can be used as a point name. This means that any integer offset (converted to ASCII) from 0 to 65535 can be used as a point name for a Mark V controller. If there is a point configured for that offset then the GSM server returns the appropriate number of bytes. If no point is configured for that offset then two bytes are returned to the client and the point is assumed to be a 16-bit integer.

Note It is not recommended that the customer use this point name system.

7 Glossary of Terms

ACK Acknowledge message
Alarm dump The alarm dump message requests a list of all active alarms.
ASCII American Standard Code for Information Interchange is an 8-bit code used for data.
ASCIIC An string of ASCII characters preceeded by a single byte integer count of the number of characters to follow, also known as a counted-ASCII string.
ASCIIZ A string of ASCII characters terminated by a zero byte, also known as a NULL terminated string.
Command A message, usually initiated by the operator, calling for a change in the controller settings, or requesting specific controller data.
CSDB Control Signal Database contains all controller signals located in the HMI.
DCS Distributed Control System is a digital process control system used for plant process control applications, including power plant control.
Gateway The software in the HMI server that controls communication between the DCS and controllers.
GMT Greenwich Mean Time, a global time standard.
GSM GE Standard Messages, used for communication with the DCS.
Heartbeat A message from the DCS to the HMI gateway indicating that the DCS is functioning.
Little endian The byte order used by many microprocessors. For multi-byte data, the least significant byte is transmitted first while the most significant byte is transmitted last.
Long name The long name is the text description associated with a point name.
Pushbutton command A pushbutton command causes the pulse of a logic signal inside a controller. Many GE controllers allow the sender of a pushbutton command to specify the duration of the pulse.
Setpoint command A setpoint command sets a value inside a controller. This is typically an analog value, but it can be any type that the controller supports.
SOE Sequence of Events - a record of contact closures and events, usually taken at a one ms rate for later investigation of turbine generator trips.
TCI Turbine Control Interface is the GE supplied software package in the HMI server that interfaces to the turbine control.
TCP Transmission Communication Protocol is used on Unix and Ethernet communication networks to control data transfer.
Telnet Software used to examine data in the server and trace messages passed to and from a client.