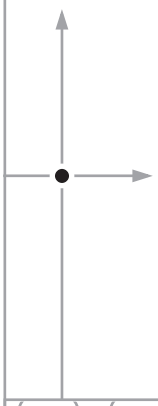
grass valley

Routing Products

Protocols Manual



071020103
MARCH 2004

the most watched worldwide

Contacting Grass Valley

Region	Voice	Fax	Address	Web Site
North America	(800) 547-8949 Support: 530-478-4148	Sales: (530) 478-3347 Support: (530) 478-3181	Grass Valley P.O. Box 599000 Nevada City, CA 95959-7900 USA	www.thomsongrassvalley.com
Pacific Operations	+852-2585-6688 Support: 852-2585-6579	+852-2802-2996		
U.K., Asia, Middle East	+44 1753 218 777	+44 1753 218 757		
France	+33 1 45 29 73 00			
Germany, Europe	+49 6150 104 782	+49 6150 104 223		

Grass Valley Web Site

The www.thomsongrassvalley.com web site offers the following:

Online User Documentation — Current versions of product catalogs, brochures, data sheets, ordering guides, planning guides, manuals, and release notes in .pdf format can be downloaded.

FAQ Database — Solutions to problems and troubleshooting efforts can be found by searching our Frequently Asked Questions (FAQ) database.

Software Downloads — Software updates, drivers, and patches can be downloaded.

Contents

Preface

About This Manual

This document describes external protocols for Grass Valley Routing products. It assumes basic familiarity with routing equipment. For information on Grass Valley Routers refer to product manuals.

Section 1

Protocols Overview

Introduction

The Grass Valley Router System can be controlled by external devices, and can also control external devices. Several methods are employed to accomplish this control. This manual covers the following protocols that allow external control of part or all of a Grass Valley Router System:

- *Grass Valley Router Control Language* on page 11 (Encore Control System Protocol via RS-232, RS-422, SLIP, or Ethernet),
- *Series 7000 Native Protocol* on page 61 (Series 7000 Signal Management System Protocol via RS-232, RS-422, SLIP, or Ethernet), and
- *Serial Node Controller Protocol on page 103* (Pro-Bel Protocol via RS-485).

  Omnibus automation system control of a System 7000, using a variation of the Pro-Bel protocol.

Other control mechanisms listed below that involve protocols are described elsewhere:

- Encore Diagnostic (via VT-100 terminal or emulator),

  For reporting diagnostic information, and for issuing simple commands for diagnostics and service purposes. This interface is described in the *Encore Installation and Service Manual*.
- System Diagnostic Interface (Series 70000 via VT-100 terminal or emulator),

  For reporting diagnostic information, and for issuing simple commands for diagnostics, service, and installation purposes. This interface is described in the *Series 7000 Installation Manual*.
- Pro-Bel System 3 Interface, and

  System 7000 Control of a Pro-Bel System 3. This interface is described in the *Series 7000 Installation Manual*.
- Networked Series 7000 systems.

  Uses Native Protocol. Issues relating to networked control are described in the *Series 7000 Configuration Manual*.

General Interface Requirements

For successful control, all devices involved must be physically connected with the proper interface, have any required options installed, and be properly configured for the desired connection and protocol.

Requirements differ for different control mechanisms, and are described where appropriate.

Refer to the manufacturer's documentation of any external device being interfaced to a Grass Valley Router System for information on installation and configuration procedure for these systems

Section 2

Grass Valley Router Control Language

Introduction

The Grass Valley Router System can be controlled by an external, serial or ethernet connected communicating device such as a personal computer or an automation system. The Router Control Language (RCL) described is intended to facilitate control of the Grass Valley Router.

Commands and error responses are terse and character efficient to maximize throughput. All message bytes are from the ASCII character set (printable and control characters). This provides the ability to easily log information through the duplex control ports.

Message sizes can be tuned to match receive buffer sizes to maximize throughput.

RCL is implemented in Grass Valley Encore routing systems from Encore Software Version 1.6.5 onwards. SMS7000 and earlier Encore systems do not support RCL.

RCL protocol offers significant advantages over its predecessor Native protocol. The following features facilitate the client to control the routing system in a better way:

- *Synchronizing Requests and Responses* on page 26,
- *Multiple Area Support* on page 26,
- *Subscription Support* on page 27,
- *Exclusion Set Support* on page 28, and
- *Refreshing and Maintaining Protects* on page 28.

Command and Message Description Notation

For the command parameter descriptions in this section, lower case parameters must be replaced by user specified information, while upper case parameters must be literally supplied.

Upper case parameters fall into two categories: printable ASCII characters, where they are supplied as shown, and ASCII control characters, where the text shown translates into a hex equivalent.

Notation symbols used in the format descriptions in this section are shown in Table 1.

Table 1. Notation symbols

Symbol	Meaning
...	A continued sequence.
|	Or
[]	Optional parameters
< >	Choices, or ASCII control characters, or for clarity.
,	Comma designates horizontal tab <HT>, the data separator.

For the sake of readability, spaces may be shown in the descriptions where none exist in the protocol definition.

Interface Requirements

In order to control a Grass Valley Router system using RCL, both the Grass Valley Router and the external device must have RCL protocol implementation.

Communication with the Grass Valley Routing system can be achieved using either a RS-232, a RS-422, or an Ethernet interface.

RS-232 or 422 Communication

Pinouts and cable diagrams for creating RS-232 or 422 connections are available as part of the Installation instructions in Grass Valley product manuals.

Default communication settings are:

- 9600 Baud
- 8 Data Bits
- 1 Stop Bit
- No Parity

The external device controlling the Routing System must be equipped with an RS-232 or RS-422 Serial Port capable of supporting at least one of the following communication rates; 300 k, 600 k, 1.2 k, 2.4 k, 9.6 k, 19.2 k, 38.4 k, or 115.2 k Baud.

Ethernet Communication

Use of Ethernet is recommended. The network should be closed, for use only by the Grass Valley Router system. Configurations that are connected to larger, open networks, are not supported.

Each device on the network must be assigned a unique IP address and name. Review the instructions for Ethernet interface hardware in your computer manuals. It may be necessary to consult an expert in the field of Ethernet network installation.

To use the RCL Ethernet interface, user-supplied software must be created to send and receive RCL messages according to the protocol specifications in this document. The programmer writing software for this application must be skilled in the use of TCP/IP sockets. Knowledge of Ethernet networking and system administration is also required to install and configure software.

For Grass Valley Router systems which do not support Ethernet, external control using RCL control is accomplished via a RS-232 or RS-422 interface.

Basic RCL Description

The levels of the RCL Protocol are defined in Table 2:

Table 2. Protocol Levels

Level	Description
Level 1	Physical (e.g. RS-232, RS-422, Ethernet)
Level 2	Data Link (e.g. checksums, ACK/NAK)
Level 3	Supervisory (e.g. flow control, message buffering)
Level 4	Application

The following discussions assume that Levels 1, 2, 3, and 4 will be similarly implemented on each end of the communication link. Because of significant differences in all but level 4 messages, the RS-232/RS-422 and Ethernet descriptions are presented separately.

RS-232 and RS-422 Description

Level 1

- RS-232 or RS-422
- Baud rate - 300 k, 600 k, 1.2 k, 2.4 k, 9.6 k, 19.2 k, 38.4 k, or 115.2 k (Default = 9.6 k)
- 1 stop bit
- 8 data bits
- No parity

Level 2

Level 2 adds the <SOH> character and the protocol_id to the message byte stream. It calculates the message checksum and appends it to the message. It then adds the <EOT> character, and transmits the message.

The receiving end, buffers input, verifies the <SOH>, protocol_id, and <EOT> bytes, and verifies the checksum. If the message is successfully received, it notifies Level 3 of its availability.

Level 2 ACK/NAK

When a message is successfully received by the router, an ACK (0x06) message is returned to the client. The client also should send an ACK when it receives a complete message. ACKs are returned immediately, with no field delay. ACKs are not encapsulated in <SOH>, <checksum>, or <EOT>. If the sender does not receive an ACK within 500-1000 milliseconds after the message is sent, the message is re-transmitted for a total of ten (10) attempts. The Router end of the protocol will always attempt to transmit exactly ten (10) times. However, the external device may choose fewer attempts, or simply keep transmitting the same message until it is finally acknowledged.

If an error occurs during reception of a message, a NAK (0x15) followed by an error code descriptor is returned to the sender. NAKs are returned immediately, without any delay. NAKs and Level 2 error codes are not preceded by an <SOH> or verified with a <checksum>. However an <EOT> trails the NAK message, as follows:

```
<NAK> <ErrorCodeHI> <ErrorCodeLO> <EOT>
```

The error code is a two digit hex number, expressed in ASCII.

The sender can attempt to re-transmit. However, the sender should not attempt transmission of a new or repeated message until an ACK or NAK is received, or until an appropriate time-out occurs. If a NAK with an error code indicating buffer not available is received, the sender should delay before attempting a re-transmission.

When the external device returns NAKs to the RCL Server, they must be in the format described above (with error code and EOT).

See *Level 2 (NAK) Error Code Descriptions* on page 25 for specific error information.

Level 3

Level 3 copies input messages from Level 2 buffers into its own buffers. By marking the Level 2 buffers as available, it effectively accomplishes flow control (assuming the sender delays long enough before attempting to retransmit the message, and the receiver gets a buffer cleared out in time). Should buffers become full, Level 2 will return the required <NAK> <buffer not available error> <EOT>, for every message attempted while this buffer (or queue) full condition remains. Again, a delay between transmitted messages should be invoked when this error condition is reported to the sender.

Level 3 passes incoming data buffers up to the appropriate Level 4 protocol handler one at a time. Level 3 appends the RCL Server internal header to the front of the RCL message before delivering to Level 4.

Level 3 receives output messages from Level 4, one at a time, and buffers them for output to Level 2. Level 3 strips the RCL Server internal header from the message, only passing on to Level 2 the properly formatted RCL message. Messages are passed to Level 2 one at a time. Level 2 calculates the checksum and transmits.

Level 3 Error Recovery

Ten (10) consecutive transmit retries of a message will be attempted. Should the packet not be accepted by the receiving side within that count, an error will be sent. Appropriate action could be a switch over to the redundant Interface Card if redundant pairs are involved, or an Interface reset in stand-alone configurations.

In both transmit and receive cases, the retry count is reset to zero on receipt of the ACK.

See *Error Codes* on page 24 for specific error information.

Level 4

Level 4 of RCL parses the content of the messages, and accesses the Grass Valley Router to either return the information requested, or to perform the requested action.

See *RCL Message Categories* on page 30 for a detailed listing of all Level 4 messages.

Ethernet Description

Level 1 Description

- 10-Base2 Ethernet
- Closed network strongly recommended (dedicated to Grass Valley Router)

Levels 2, 3, and 4 Description

There are two widely used socket types, stream socket, and datagram socket. Each uses its own communications protocol. Datagram sockets use UDP (User Datagram Protocol), which is a message oriented protocol. Stream sockets use TCP (Transmission Control Protocol), which is a stream oriented protocol.

Datagram sockets have to read entire messages at once and carry sufficient information to be routed from the source device to the destination device. They do not rely on earlier exchanges between the source device, destination device, and the transporting network.

Stream sockets (sometimes referred to as Berkeley sockets) treat communications as a continuous stream of characters and are connection oriented. Therefore, a connection must be opened and maintained for the duration of the communications. Stream sockets are supported for many different host environments and operating systems.

A computer or other host device using RCL must initiate communication with a Grass Valley router. The computer is a client, and the Grass Valley router is the server.

RCL Session Sequence

A RCL client can establish connection to RCL Server and perform operations using the steps mentioned below.

1. Create a TCP/IP stream socket on the client.
2. Set linger on with a timeout of zero.
3. Connect the socket to the IP address of the desired RCL Server, on port 12345.
4. Perform RCL Connect (RC) sequence:

   a. Request RCL connect to the RCL Server

   ```
   RC
   ```

   b. Response

   ```
   RA,session_id
   ```

5. Do the necessary queries and operations on the router.

6. Once the client finishes the operations and no longer wants to communicate with the router it can disconnect by initiating the RCL disconnect (RD) as described in page 8.

7. Close the TCP socket that was created during the initiate communication phase.

Data Link and Supervisory Control

Unlike RCL via Serial, the TCP/IP communications layers used with Ethernet are responsible for the end-to-end error-free transport of messages. This means that messages sent and received via stream sockets are guaranteed to arrive in order and error-free, as long as the connection between the client and server is maintained.

In RCL via an Serial, Levels 2 and 3 are responsible for the error-free transport of messages. Since data transport is managed transparently for TCP/IP stream sockets, the Level 2 ACK/NAK protocol is not used for Ethernet communications. The user-supplied software must not generate or expect ACKs or NAKs for message transactions. Level 2 error messages will not be generated.

Since the TCP stream connection is error-free, message format and checksum errors are generally a result of a programming error and not a communications error. The user-supplied program should be able to prevent these errors.

Sending Messages

While connected, RCL Level 4 messages may be sent from the client by writing to the socket. Each message sent must be properly constructed as documented for RCL messages.

The requirements for each message are:

- The message must begin with a SOH,
- End with an EOT, and
- The transmitted checksum must be correct.

The user-supplied software must check for errors returned by the socket's write function call to ensure that the entire message was accepted and transmitted correctly.

All RCL Level 4 messages and responses are available to an Ethernet client. However, since the Level 2 ACK/NAK is not used, the `BK,2` command will be ignored (no response is generated).

One possible source of problems is in the checksum verification. Message checksums as defined for Level 2 must be calculated and included with all messages. The RCL Server interface will discard any message with a

checksum error, and a Level 2 error message will not be returned. Since TCP/IP guarantees an error-free message, the checksum cannot be corrupted during transmission. However, if the user-supplied software incorrectly calculates the checksum, the message will not be processed by the Server.

With stream sockets, the user-supplied software must correctly handle byte ordering and padding for multi-byte values. However, all RCL messages are comprised of single byte values (ASCII characters), so byte ordering is not a problem when the correct message format is followed.

Receiving Messages

RCL Level 4 responses are received by the client by reading from the connected stream socket. Each message is formatted as documented, beginning with a SOH character and ending with an EOT character.

When reading a TCP/IP stream socket, data is presented error-free and in the order sent. However, there is no built-in method for identifying the boundaries of messages. It is up to the user-supplied software to look for the beginning SOH and ending EOT. Be aware that most stream socket implementations may deliver message fragments when the read function call is made. The user-supplied software must be designed to buffer received messages until a complete message is received. Response messages will be received for all command messages sent to the server.

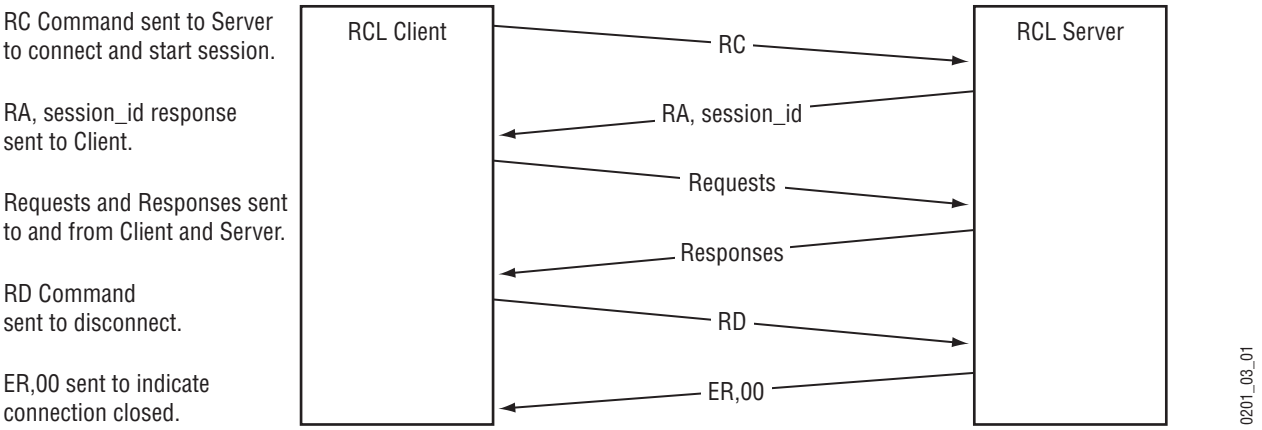
RCL Connection Management

RCL Connect

The TCP or serial connection mechanism is followed by an application level RCL connect mechanism, before the client can do any other transaction with the Grass Valley Router. The RCL connect sequence is.

1. The client will send a connect request command `RC` (described in *RC - RCL Connect* on page 47).
2. The server will respond back with RCL accept message `RA` (described in *RC - RCL Connect* on page 47). The server response `RA` will have an unique `session _id` for this connection.

Figure 1. RCL Client and Server Communication



RCL Disconnect

Either the client or server can initiate the RCL disconnect process by issuing the `RD` command (described in *RD - RCL Disconnect* on page 47). This is the application level disconnect which needs to be done before a socket closure. The end that receives disconnect request will respond back with an `ER, 00` and the connection will be closed gracefully.

This process is depicted in the diagram above. This shows the RCL connect and disconnect mechanism and the flow of commands and responses during this process.

RCL Announce

The RCL server during start up will send a command called RCL Announce (`RN`) to all the connected clients indicating that the server has just started up. The format of the command is: `RN`

RCL Message Format

Every RCL message has a fixed length header, a variable data field and a fixed length trailer.The size of the data field can be between 0 and 1007 bytes. The fields in the RCL message is case sensitive. RCL request and response message format is described below.

```
<SOH> <protocol_id> <session_id><message_id>
<seq_number><Reserved> <Data> <checksum> <EOT>
```

The significance and the valid values of the various fields in the message is shown in Table 3.

Table 3. RCL Message Format Fields

Field	Description
<SOH>	ASCII Start of Header character (Value = ASCII 01)
<protocol_id>	The protocol identifier (one ASCII character) (Value = 'R')
<session_id>	Two hex digits converted to its ASCII representation. The server at session initiation will give the session ID.
<message_id>	Four hex digits converted to ASCII representation. The unique ID of the message in this session.
<seq_number>	Four hex digits converted to ASCII representation. The most significant byte is sent first. This indicates the number of the message in its sequence. 0000 if it is the last message.
<Reserved>	Four hex digits reserved for future use. To be filled with value 0xFFFF.
<Data>	The request / response message. The parameter delimiter is <HT> Horizontal Tab (0x09), and precedes each parameter. (Note that in the examples, A comma is used to signify <HT>.).A trailing <HT> following the last datum is essential to facilitate parsing. The data is of variable length and depends on the command \ response. The maximum length of this field can be 1007 bytes.
<checksum>	Two-byte check sum value for error checking. Refer to the checksum calculation algorithm section below.
<EOT>	ASCII End of transmission (Value = ASCII 04).

Table 4 is an example showing how the various fields in the RCL message should be used. The `BK,D` command is used in the example below.

Table 4. Example of BK,D Command

Field	Byte	Byte in ASCII (as sent in message)	Length (in bytes)
SOH	SOH	01	1
protocol_id	R	52	1
session_id	4 5	34 35	2
message_id	1 2 3 4	31 32 33 34	4
seq_number	0 0 0 0	30 30 30 30	4
Reserved	F F F F	46 46 46 46	4
request_cmd	B K	42 4b	2
HT	HT	09	1
parameter	D	44	1
checksum 0	TBD	TBD	1
checksum 1	TBD	TBD	1
EOT	EOT	04	1

Checksum Calculation Algorithm

The checksum is calculated on those items following `SOH` and before the inserted checksum value. All the values between `SOH` and the checksum field are summed up and mod 256 on this is calculated. This is then negated to arrive at the two byte checksum.

Please refer to *Appendix A-CheckSum Calculation Code Snippet* page 113 for the code snippet showing the checksum calculation for the `BK,D` example from Table 4.

52+34+35+31+32+33+34+30+30+30+30+46+46+46+46+42+4b+09+44=437

Mod 0xFF of 437 = 37

To negate that value: (+ff)+(-37)+(+01)=C9

The checksum value is converted to two hex digits and inserted in the message as shown in Table 5

Table 5. Checksum Value Conversion to Byte

Field	Byte	Byte in Hex (as sent in message)	Length (in bytes)
SOH	SOH	01	1
protocol_id	R	52	1
session_id	4 5	34 35	2
message_id	1 2 3 4	31 32 33 34	4
seq_number	0 0 0 0	30 30 30 30	4
Reserved	F F F F	46 46 46 46	4
request_cmd	B K	42 4b	2
HT	HT	09	1
parameter	D	44	1
checksum 0	C	43	1
checksum 1	9	39	1
EOT	EOT	04	1

Responses and Errors

As soon as the RCL server receives the request from the client, it processes the request and sends a positive or negative response based on the status of the performed operation. The maximum time the client will have to wait for a response from the server is called the Latency Time. This Latency time is the timeout period for the client to retransmit requests if it did not get the response.

The default mode of operation would include response for all commands. The clients can enable/disable responses for operations in order to reduce processing at the client end. The client can disable these responses by sending the RCL command `BK,E,OFF`.

In case of operations like take, protect, unprotect and salvo, positive response only indicates that the operation request has been passed onto the lower layers in the system. It does not indicate successful completion of the operation. The only way to find out if the operation has been successfully carried out is to query the status back on the specific entity on which the operation was performed. In case of clients who have subscribed for status changes on the entity, the change will be asynchronously notified.

Level 4 Error

The RCL level 4 error message is an RCL message with the following data. The header and trailer information is the same as any other RCL message.

```
<SOH><protocol_id><session_id><message_id><seq_number>
<Reserved><Data><checksum><EOT>
```

Where `<Data>` is of the form `ER<,error_code><,request_cmd>`

`ER`	The two ASCII characters 'ER'
`<,error_code>`	Two digit ASCII code defining the error detected at Level 4. Level 4 error codes are two digit hex numbers, and are transmitted as two ASCII bytes, most significant byte first. "ER,00" signifies that the request was successfully executed and is not an error.
`<,request_cmd>`	The two-letter request command which this message corresponds to.
`[,data]`	Optional printable ASCII character providing additional information about the error. In many cases, this data will be one of the parameters of the failed command (e.g., the incorrect dest_name). The datum delimiter is <HT> Horizontal Tab (0x09), and precedes each datum, including the first. (Examples show <HT> as comma.) A trailing <HT> following the last datum is sent to facilitate parsing by the client. Incoming control messages are buffered to maximize throughput.

Message Size and Sequence

Message size can grow quite large, both for commands and responses. For example, the `TA` (Request Take) command could grow large, depending on the number of sources and levels specified. Long commands and responses may need to be segmented and sent in a sequence of messages rather than in one large message.

The seq_number indicates the sequence of this packet in the context of the whole message.

If <`seq_flag = ASCII '0000'`>, this is the last segment.

Many messages fit in a single packet. For these messages also, the sequence number is set to `'0000'`.

Messages are sent with entries intact, so each message makes complete sense on its own. Messages are not arbitrarily broken up without regard to data boundaries.

Level Bitmap

A level_bitmap is a 32 bit quantity where each bit represents the presence (=1) or absence (=0) of a particular level for that command or response. The least significant bit (right-most) represents Level #0. The most significant bit (left-most) represents Level #31. The `QN,L` command allows the user to find out the Level Names configured in the system.

The following example shows how level bitmap is constructed for a given set of levels.

Assume that a particular operation is to be performed on levels 0, 2, 3, 4, 5, 6, and 10. This information is rendered onto a 32 bit quantity as shown below.

Level Bitmap = `0000 0000 0000 0000 0000 0100 0111 1101` (Binary)

The equivalent hex message is shown below

Level Bitmap (Hex) = `0x 0000047D`

These 8 hex digits are then converted to ASCII (='0',...'9', 'A',...'F') and sent with the most significant byte first. The hex digits 'A'...'F' can be sent as upper or lower case ('a'...'f').

Level Bitmap (Sent in ASCII) = `30 30 30 30 30 34 37 44`

Area Bitmap

An area_bitmap is a 64 bit quantity where each bit represents the presence (=1) or absence (=0) of a particular area for that command or response. The least significant bit (right-most) represents area index #0; the most significant bit (left-most) represents area index #63. The `QN,A` command allows the user to find out the Area Names configured in the system.

The following example shows how an area bitmap is constructed for a given set of areas.

Assume that a particular operation is to be performed on areas 8, 13, 19, 24, 25, 28, and 32. This information is rendered onto a 64 bit quantity as shown:

Area Bitmap = `0000 0000 0000 0000 0000 0000 0000 0001` (Bits 63-32)
(Binary) `0001 0011 0000 1000 0010 0001 0000 0000` (Bits 31-0)

The equivalent hex message is:

Area Bitmap = `0x 00000001 13082100`
(Hex)

These 8 hex digits are then converted to ASCII (='0',...'9', 'A',...'F') and sent with the most significant byte first. The hex digits 'A'...'F' can be sent as upper or lower case ('a'...'f').

Area Bitmap = `30 30 30 30 30 30 30 31 31 33 30 38 32 31 30 30`
(Sent in ASCII)

Error Codes

RCL server may return Level 2 NAK errors or Level 4 errors in case of request failures.

Level 2 NAK Errors

Negative Acknowledgement (NAK) is generated due to communication related failures to a serial (RS-232 or RS-422 interface) client. These errors are not present for the Ethernet RCL interface.

An example of a Level 2 error is a Time Out Error. Time out interval begins upon reception of an `SOH` and is halted at the reception of an `EOT`. Time out interval is one (1) second for data rates from 2400 to 38.4k Baud.

For slower rates, time out is calculated as:

Time out (in seconds) = 2400/data rate.

For example, at 300 Baud, time out = 8 seconds (2400/300 = 8).

Level 2 (NAK) Error Code Descriptions

The following codes are sent with NAKs from the Router to the RCL client. The client is also responsible for sending NAKs to the Router when appropriate. However, if specific errors are reported with an external device's NAK, they should be defined as:

`<NAK> <error_code> <EOT>`

`<error code>` is defined as a Hexadecimal number 71 - 79 (Table 6).

Table 6. RCL Protocol Level 2 NAK Error Codes

Decimal Value	Hexadecimal Value	Meaning	Description
113	71	Buffer Size Exceeded	The number of characters received since the last detected SOH is greater than the maximum RCL message length.
114	72	Buffer Not Available	Input buffer full.
115	73	Reserved	
116	74	Chip Level Error	Error detected by UART such as parity error.
117	75	Checksum Error	Packet had a bad checksum. Low Level errors such as framing, overrun, etc., are reported as checksum errors.
118	76	Time Out Error	Time out interval is begun upon the reception of an SOH, and is halted at the reception of an EOT. Time out interval is one (1) second for data rates from 2400 to 38.4k Baud. For slower rates, time out is equal to 2400/data rate = time out in seconds. For example, at 300 Baud, time out = 8 seconds (2400/300 = 8).
119	77	Missing SOH	EOT is detected without a preceding SOH.
120	78	Missing EOT	No EOT detected in message.
121	79	Reserved	

Level 4 Errors

RCL Level 4 Errors occur when the commands are parsed and processed. Refer to *Appendix B-Level 4 Error Codes* page 115 for a list of Level 4 Error Codes.

A client can retrieve an explanation of the numeric error code at run time by using either the All Level 4 Errors or the Specific Level 4 Errors retrieval method.

All Level 4 Errors Retrieval

The client can request a list of all Level 4 error codes and definitions using the command `QE`.

The Router will respond with separate messages, each containing an error code and a description. Copy or print the error listing so that the explanations are at hand when error codes are received.

Specific Level 4 Errors Retrieval

Use the `QE` command with specific error code parameters.

RCL Features

Synchronizing Requests and Responses

Message and session IDs are provided to facilitate the clients, to match RCL requests with the responses that are obtained from the server.

The client has to generate a message ID for every request it sends to the server. The client can use the message ID to match the response with request in a particular session. The RCL server copies the message ID (received as part of the request) onto the response. This mechanism makes it possible to associate the responses with the requests that triggered them, even in case of multiple responses. It must be noted here that message ID's may wraparound in a session. Message IDs can range from 1 - to - 65535. This large range ensures that there will be no ambiguity in the response matching due to wraparound of message IDs. Message ID with value 0 is reserved for server use. The server will use this when it has to send error response and has not been able to extract a valid message ID. Notifications sent as part of subscription response will have its own sequence of message IDs with respect to a session.

Each RCL session is identified uniquely by a session ID. This session ID will be provided by the RCL server during the session initialization phase as described in the RCL connect mechanism. The client will have to use this session ID for any further request to the server. The server will copy the same as part of the response. Session ID value 0 is reserved. The RCL client will use this when it issues a RC. Since the session ID is given back to the client as part of response to RC, the client will need to use this default value for RC command.

Multiple Area Support

Areas create hierarchies within the control system and make it easier to group sources and destinations in a large system. Once an area is defined the sources and destinations in the area are identified using fully qualified names and indices. A source or destination name is said to be fully qualified if it is prefixed by "area name:". Similarly source or destination indices need to be prefixed with "area index:" to make them fully qualified. Some RCL commands also take an optional parameter called Area Bitmap to specify a set of areas on which a particular request is intended for. Please refer to the *Area Bitmap* on page 24.

Subscription Support

Subscription is a mechanism through which RCL clients can register for notifications regarding configuration and status information. Whenever these parameters change, the client is notified asynchronously by the RCL server. RCL facilities for subscription can be described in terms of the following categories.

- Status change subscriptions
- Configuration change subscriptions

The subscription sequence and the commands used for this are introduced below. The command syntax and response is described in *Client Subscription Message Descriptions* on page 53. The way these commands (and their parameters) are used is explained in *Subscription Commands Usage* on page 55.

- The client after establishing connection with the server can subscribe for any of the information. The Subscribe (`SB`) command should be used for this support and is described in *SB – Subscribe* on page 53. Once the subscription request is received, the server confirms the subscription registration by issuing an `ER,00` command. If the subscription request fails, an appropriate `ER,nn` message is generated. (Where nn is the two-digit error code sent back by the server identifying the cause of this failure.)
- The client can at any point decide to unsubscribe for a particular request that it had subscribed earlier. The Unsubscribe (`UB`) command should be used for this and is described in *UB – Unsubscribe Request* on page 53. When an unsubscribe request is received the server confirms the un-subscription by issuing an `ER,00` command. If the unsubscribe request fails, an appropriate `ER,nn` message is generated. (Where nn is the two-digit error code sent back by the server identifying the cause of this failure.)
- The server sends an asynchronous notification (`NY`) to the client whenever the subscribed status/configuration information changes are reported in the system.

The RCL server retains the subscriptions for each client till the client is connected. However, if a refresh rate is configured for the RCL client through the RCL Client Configuration screen of the OUI, then the subscriptions are retained only as long as the RCL Client refreshes the connection within the refresh timeout period specified. A refresh rate of zero means that the subscriptions are retained as long as the client remains connected. The subscriptions for all the clients are dropped in case of server reboot. In this case, RCL server will send a command called RCL Announce (`RN`) to the connected clients indicating that the client should resend the subscriptions to continue receiving the notifications.

Exclusion Set Support

Note Level exclusion sets are supported. However, area and destination exclusion set support is not implemented for Encore Software Version 1.6.5, which is the current software release at the time of this manual's revision. Check the Encore Release Notes for software versions released after Version 1.6.5 for the announcement of implementation.

The mechanism of exclusion sets (area, destination and level exclusion sets) can be applied to RCL clients. The exclusion sets to be used by any RCL client is configured at the system level (outside the scope of RCL). Any destination, area or level that is excluded for a particular RCL client is not visible to it through any operation. For example, when a client queries for the list of destinations, those destinations that are excluded are not returned.

Whenever the exclusion set changes, if the client has subscribed for notification, the client is informed about the change and needs to reinitialize its operational set of sources and destinations. These are intimated as configuration change notification. The `BK,F` flags described in Table 12 will also be set to reflect the changes in the configured names applicable to the client due to changed exclusion sets. The clients can choose to poll it at any point to know what has changed and get the information accordingly.

Refreshing and Maintaining Protects

If an automation client protects particular destinations on the Router, the client is responsible for refreshing those protects periodically or the protects will be dropped by the RCL Server.

The refresh interval can be disabled (= 0), or set ≤ 255 seconds. If any Level 4 command is not received within this periodicity, the RCL Server decides that the external device is no longer active and sends a device-delete message to the system. As a result, all protects (and Subscriptions) currently held by this automation client are dropped.

The refresh interval can be set from the client config screen of the OUI. The `BK`, with no parameters, has no side effects and can be used to keep protects refreshed in the absence of other Level 4 command activity. Level 2 ACKs from the external automation clients do not refresh the protects.

Protects issued by a client are also dropped when the RCL Server loses the connection with the client. In case of serial clients, if the connection is not closed gracefully (using `RD` command) protects remain till the next connect request from the same client. These are cleared during the next RCL connect (using `RC` command) from the client.

The configuration flags on the user interface with respect to protect function are:

Maintain protect persistence	When this flag is set, any protects performed by the client will be maintained across sessions of the same client. If this is not set, protects that a client has issued will be dropped when connection is lost with the client or when the client fails to refresh the protects within the configured time interval.
Protect Override	When this flag is set for a client, the client can override the protect that had been performed on a destination by a different control point and modify the destination status. For example if client A has protected destination Mydest, another client B with the protect override set can perform a take onto the same destination.

RCL Message Categories

Client Originated Messages

The RCL Protocol messages originating from a client are in Table 7.

Note The Chop (CH) and the Take Monitor (TM) commands are not implemented for Encore Software Version 1.6.5, which is the current software release at the time of this manual's revision. Also, the Protect (PR) and Unprotect (UP) commands are implemented for All Level Protects but not specified levels. The Query Names (QN)command will not return TieLine parameters. Check the Encore Release Notes for software versions released after Version 1.6.5 for the announcement of RCL command implementation.

Table 7. Client Originated Messages

Command	Description	Expected Server Response
BK - Background Activities (page 33)	Query system configuration and flags	KB
CH - Request Chop (page 35)	Initiates chopping between specified sources.	ER, 00
GT – Get Current VITC Time (page 36)	Get the current VITC time from the server in HHMMSSFF format as ASCII characters.	TG
PR - Protect (page 36)	Protects a specific destination from other control points	ER, 00
QC - Query Combined Destination Status (page 37)	Queries source status on combined levels of the destination. The combined levels are interpreted with respect to the first level on which the destination is present. The status returned back will have the source taken to the destination on the first level on which the destination is present. This will also specify the other levels on this source has been taken to.	CQ
QD - Query Destination Status (page 38)	Queries sources assigned to destinations by destination name.	DQ
QE - Query Error Definition (page 40)	Queries text describing a particular error code.	EQ
QI - Query Destination Status on a Specific Level by Index (page 40)	Queries sources assigned to destinations by Destination index and Level Index.	IQ
QJ - Query Destination Status by Index (page 41)	Queries sources assigned to destinations by Destination Index for all levels.	JQ
QN - Query Names (page 43)	Checks names associated with Sources, Destinations, Levels, Salvos, Areas or TieLines.	NQ
QV - Query Salvo Elements (page 46)	Queries sources, destinations, and levels associated with a specified Salvo.	VQ
RC - RCL Connect (page 47)	Request RCL connect to the server.	RA
RD - RCL Disconnect (page 47)	Request RCL Disconnect.	ER, 00
TA – Take (Breakaway) (page 48)	Takes Sources (on specified levels) to specified destination, by name.	ER, 00
TD - Take (Single Source) (page 49)	Takes a single source to all or specified levels.	ER, 00
TI - Take by Level Index (page 49)	Takes same source (on all or specified level) to specified destination, by index.	ER, 00
TJ - Take by Level Bitmap (page 50)	Takes Sources (on specified levels) to specified destination by level bit map. Allows Breakaways. More than one source can be specified to be taken on to the destination on different levels.	ER, 00
TM - Take Monitor (page 51)	Takes Destination (on specified levels) to the Monitor.	ER, 00
TS - Take Salvo (page 52)	Executes the specified Salvo.	ER, 00
UP - Request Unprotect (page 52)	Removes previously applied Protect from specified Destination.	ER, 00

Client Subscription Messages

The Subscription messages originated from the client are in Table 8.

Table 8. Client Subscription Messages

Command	Description	Expected Server Response
SB – Subscribe (page 53)	Subscribe for status/configuration changes.	ER, 00
UB – Unsubscribe Request (page 53)	Unsubscribe for already subscribed status/configuration information	ER, 00

Server Originated Messages

The RCL Protocol messages originated from the server are in Table 9.

Table 9. Server Originated Messages

Command	Description	Expected Client Response
NY – Notification (page 54)	An asynchronous Notification sent whenever the subscribed status/configuration information changes	ER, 00
RD - RCL Disconnect (page 54)	Request RCL Disconnect.	ER, 00
RN - RCL Announce (page 55)	Server announce at start up to facilitate clients to refresh subscription requests.	None.

RCL Message Specifications

The following specifications apply to all the RCL message commands.

Note Leading zeros are used if necessary so as to keep the area index length to 2 digits and destination/source index length to four digits.

`fullqual_dest_name` Specifies the fully qualified destination name. This is of the form `area_name:destination_name`.

Example, for a destination called `GRAFIX` in area `NEWS`, the fullqual_dest_name is `NEWS:GRAFIX.`

`fullqual_src_name` Specifies the fully qualified source name. This is of the form `area_name:source_name`.

Example, for a source called `VTR2` in area `SPORTS`, the fullqual_src_name is `SPORTS:VTR2.`

`fullqual_dest_index` Specifies the fully qualified destination index. This is of the form `area_index:destination_index`.

Example, for a destination with index 127 in area with index 2, the fullqual_dest_index is `02:007F`. (Use the hexadecimal values for index 127 which is 00F7 and index 2 which is 02.)

`fullqual_src_index` Specifies the fully qualified source index. This is of the form `area_index:source_index`.

Example, for a source with index 64 in area with index 1, the fullqual_dest_index is `01:0040`. (Use the hexadecimal values for index 64 which is 0040 and index1 which is 01.)

Every RCL Client may configure a default area through the client configuration screen of the OUI. If the field `'area_name:'/ 'area_index:'` is omitted in the fully qualified name, then the default configured area is used by the RCL Server to qualify the Source/Destination name/index.

`time_stamp – VITC` Time stamp to be provided by the clients for deterministic operations. The time stamp is specified as part of the commands in `HHMMSSFF` format.

HH	Hours	00 … 23
MM	Minutes	00 … 59
SS	Seconds	00 … 59
FF	Frames	00 … 30 for NTSC 00 … 25 for PAL

In case of non time stamped operations this field can be left blank. The time stamp field is used to specify the VITC time at which the operation (take, chop or protect) should be performed on the router. The operation will be carried out exactly on the specified frame. This time should at least be 6 frames more than the time at which the command is received by the RCL Server. Thus the client should take into account the transit time (Via Ethernet or serial medium) while setting this time stamp value. There is also an upper limit on the time that can be specified by the client. The time specified should not be more than 100 frames from the time at which the RCL Server receives the request.

Client Originated Message Descriptions

BK - Background Activities

The BK command can be sent to query system configuration and flags

Command

```
BK[,parameter [,mask]]
```

One parameter can be specified per BK call. Each parameter consists of a single, case sensitive character, defined in Table 10.

Table 10. BK Command Parameters

Parameter	Description
N	Queries System Name
R	Reserved
T	Queries the software version
t	Reserved
F	Queries the configuration flags. Refer to Table 12 for the meaning of these flags.
f	Clears the configuration flags specified in the mask. Refer to Table 12 for bit definition of the mask.
D	Clears the flags associated with the QD command. After BK,D is sent, the next QD command will result in the server sending all destination statuses.
P	Queries the client specific configuration parameters specified on the graphical user interface.
E	Returns whether to send the positive responses for operations. The client can also use this command to set/stop the positive responses for operations by sending BK,E,ON or BK,E,OFF respectively.
d	Returns the name of the client specified on the graphical user interface.
2	This command is used by the serial client to ensure that the RCL server is running. The server will respond back with a level 2 acknowledgement. This command is for the serial interface only.
	No parameter, no side effects. Can be used to refresh Protects and Subscriptions

Response

For the **BK** command the response is dependent on the parameter specified. For some parameters the response will be an ER,00 to signify that the request has been executed successfully.

For other parameters response with returned data is of form:

```
KB,parameter,data
```

Parameter consists of a single, case sensitive character, defined in Table 11.

Table 11. KB Response Parameters

Parameter	Data
N	system_name_string
T	Software version string.
E	Echo = ON | OFF. A value will be returned for both set and query requests.
d	Client Name.
P	QueryOnly =<ON|OFF>[a], ChopLck=<ON|OFF>, SalvoLock=<ON|OFF>, ProtectOverride=<ON|OFF>, MonitorControl=<ON|OFF>, ControllableLevels=lvlBitMap
D	This will get an `ER,00` response on success and an `ER,nn` (nn>0) on failure indicating the reason of failure.
2	No level 4 response for this command.
F	Four ASCII characters representing four HEX digits (16 bits). These bits indicate changes of corresponding parameters since last cleared by `BK,f` command. Refer to Table 12 to associate the bits with configuration parameter. If the parameter has changed the bit is set. Most significant hex digit is sent first (that is, b15...b12). Area bitmap information will be returned indicating the areas in which source and destination name changes have occurred. This facilitates the router to reload the names only in those areas which have changed. The area bitmap is a 64 bit quantity coded as 16 Hex bytes to indicate the area 1-64. Thus the response for this parameter will be `KB, F, mask, Name change area bitmap`.
	No parameter will return an `ER,00`

[a] When a client is set as a query only device, it cannot perform any operation (Take, Protect, etc.,). It can query status and configuration information.

For F response with the returned data is of form:

```
KB, F
```

For a `KB, F` response, the data is defined in Table 12.

Table 12. KB, F Command Response Bits

Bit #	Meaning
0 (lsb)	Reserved
1	Any protect initiated by this client was dropped.
2	Destination changes
3	Reserved
4	Source changes
5	Level changes
6	Salvo changes
7	Reserved
8	Area changes
9	Client Name changes
10 - 14	Reserved
15	Reserved
f, mask	This will get an `ER,00` response on success and an `ER,nn` (nn>0) on failure indicating the reason of failure.

Example of the Use of Flag Bits

The user queries RCL using `BK,F`. They receive a reply with parameter bit #4 set to indicate that there has been a change (addition, or modification, or deletion) to the system source name table. The server will also send the source change area bit map (as part of the `KB,F` response) indicating the areas on which the source information has changed. The client should now issue a `QN,S, source_area_bitmap` command to get the new source information. The source change area bit map returned with `KB,F` can be used by the client as part of the `QN,S` command. The client next uses the `BK,f,` mask (where mask bit #4 = 1) command to clear bit #4 from the mask. The client again queries using `BK,F`. The reply shows that bit #4 = 0, indicating that the source name list just downloaded is current.

Do not confuse the flags discussed with the `f` & `F` options with those discussed with the `D` option. The `QD` command allows the user to download incremental changes in the Destination Status tables. The `D` option of the `BK` command clears the bit arrays that keep track of these incremental changes. This re-synchronizes the router data base (for Destination Status) if the external device resets.

CH - Request Chop

Note The `CH` command is not implemented for Encore Software Version 1.6.5, which is the current software release at the time of this manual's revision. Check the Encore Release Notes for software versions released after Version 1.6.5 for the announcement of RCL command implementation.

Chop command allows chop operations to be performed on a specified destination on the router. The command allows chop source to be specified on a level by level basis allowing more than one source to be specified on different levels.

Command

```
CH, fullqual_dest_name,nbr_sources,src_name_entry1
[,..., src_name_entryn]
```

(<seq_flag='0000'> for the last sequence sent.)

However, the specified source names to be taken to the destination specify the chop source names and levels. The other source for the chop operation is the already taken source on the destination. To stop chopping, issue a take on the required levels.

To specify a chop operation, first Take sources on levels to the destination, then Chop to the same destination with the chop sources on the required levels.

Response

This will get an ER,00 response on success and an ER,nn (nn>0) on failure, indicating the reason of failure.

dest_name	Destination to be taken to.
nbr_sources	Number of following entries (must be at least one).

src_name_entryn is defined as:

```
fullqual_src_name,level_bitmap
```

Example

The following command will chop a source named VTR_W_2 in area NEWS on levels with indices 2 and 3 and source VTR_M_2 in area NEWS on level index 1 to the destination VTR_B_1 in area NEWS.

```
CH,NEWS: VTR_B_1,2,NEWS: VTR_W_2,0000000C,
NEWS: VTR_M_2,00000002
```

GT – Get Current VITC Time

Get the current VITC time from the server in HHMMSSFF format as ASCII characters

Command

```
GT
```

Response

```
TG, HHMMSSFF
```

HH	= Hours	00 … 23
MM	= Minutes	00 … 59
SS	= Seconds	00 … 59
FF	= Frames	00 … 30 for NTSC 00 … 25 for PAL

PR - Protect

Note Only an All Level Protect is implemented for Encore Software Version 1.6.5, which is the current software release at the time of this manual's revision. Check the Encore Release Notes for software versions released after Version 1.6.5 for the announcement of RCL command implementation.

Protect command allows the specified destination to be protected from any source changes on the specified level. The command requires a level_bitmap to be specified on which protect should be performed.

Command

`PR,fullqual_dest_name,level_bitmap`

Response

This will get an `ER,00` response on success and an `ER,nn` (nn>0) on failure indicating the reason of failure.

Example

To protect a destination named `PDR2` in area `POSTPRD` on levels with indices 3 and 4 the client should issue the following command

`PR,POSTPRD:PDR2,00000018`

QC - Query Combined Destination Status

Query the status on combined levels of the destination. The combined levels is interpreted with respect to the first level on which the destination is present. The status returned back will have the source taken to the destination on the first level on which the destination is present. This will also specify information of the other levels on which this source has been taken to.

Command

The `QC` command can have different command syntaxes.

`QC`	In this case, destination status is returned for all destinations for which status has changed since the last time status was sent. Destination status is returned for all levels (for the changed as well as unchanged levels) which have status. The `BK,D` command can be sent before this request to force return of all destination status.
`QC[,area_bitmap:]`	In this case the destination status is returned for all destinations in the areas specified by the area bitmap for which status has changed since the last time status was sent to the client.
`QC[,fullqual_dest_name]`	In this case the status for the requested destination will be returned

Response

`CQ,`
`fullqual_dest_name,dst_level_bitmap[,src_name_entry1]`

(<seq_flag='0000'> for the last sequence sent.)

`dst_level_bitmap` Describes the levels configured for destination.

`src_name_entry1` is defined as:

```
<'N'|'P'>,<'N'|'C'>, fullqual_src_name,level_bitmap,
[prot_device_name], [fullqual_chop_src_name]
```

Parameters for this response are in Table 13.

Table 13. QC Response Parameters

Parameter	Meaning
'N' |'P'	Not-protected or Protected
'N' | 'C'	Not-chopping or Chopping
fullqual_src_name	The source currently taken to the destination on the first level on which the destination is present.
level_bitmap	Describes the levels of that destination which the source is on
prot_device_name	The device currently holding the protect. If the destination is not protected, or the device name is unknown, the field is left blank.
fullqual_chop_src_name	The name of the source chopping to this destination

If there is no source currently on the first level defined for the destination, the `fullqual_src_name` part of the `src_name_entry` will be blank in the response.

If the query is made for all the areas or all the destinations in a given area, the response will contain only those destination whose status has changed since the last query. If there are no destination status changes since the last query, the response will be a blank `CQ`. That is, no destination status will accompany the `CQ`.

`QC` (with no `dest_name` specified) is one of several commands whose response can be more than one message sequence. `QC` with no parameter can result in a sequence of messages for each of many destinations.

Example

To query the combined status of the destination `PDR2` in area `POSTPRD` the following command syntax should be used.

```
QC,POSTPRD:PDR2
```

QD - Query Destination Status

Query the destination status. This query returns status of the requested destination including the sources that have been taken to the destination along with the associated levels for the sources.

Command

QD can have different command syntaxes.

`QD`	In this case, destination status is returned for all destinations for which status has changed since the last time status was sent. Destination status is returned for all levels (for the changed as well as unchanged levels), which have status. The `BK,D` command can be sent before this request to force return of all destination status.
`QD[,fullqual_dest_name]`	In this case the status for the requested destination will be returned
`QD[,area_bitmap:]`	In this case the destination status information is returned for all destinations in the areas specified by the area bitmap for which status has changed since the last time status was sent to the client.

Response

```
DQ, fullqual_dest_name,nbr_sources
[,src_name_entry1,...,src_name_entryn]
```

(<seq_flag='0000'> for the last sequence sent.)

`nbr_sources` Number of sources on that destination which are being reported in this message sequence. If there are no sources on this destination at any of its levels, `nbr_sources = 0` will be returned.

`src_name_entryn` is defined as:

```
<'N'|'P'>,<'N'|'C'>, fullqual_src_name,level_bitmap,
[prot_device_name], [fullqual_chop_src_name]
```

Data for this response is identical to the `QC` command response described previously.

If there are no sources currently on some of the levels defined for the destination, no information is reported for those levels.

If the query is made for all the areas or all the destinations in a given area, the response will contain only those destination whose status has changed since the last query. If there are no destination status changes since the last query, the response will be a blank `DQ`. That is, no destinations status will accompany the `DQ`.

QD (with no dest_name specified) is one of several commands whose response can be more than one message sequence. QD with no parameter can result in a sequence of messages for each of many destinations.

Example

Suppose the index of area INGEST is 1 and index of POSTPROD is 5, then the corresponding area bitmap will be 0x0000000000000022

To query the status of all the destinations in areas INGEST and POST-PROD, the following command syntax should be used.

```
QD, 0000000000000022:
```

QE - Query Error Definition

Error messages returned by the server are identified by a two byte code. The QE command allows the user to retrieve the text describing Level 4 error codes. Level 2 error codes (associated with NAKs) are described in *Level 2 (NAK) Error Code Descriptions* on page 25.

Command

```
QE,[error_code]
```

`error_code` A 2 hex digit error code (to be sent in ASCII format)

Response

If the `error_code` is specified along the query, the description for that `error_code` is returned. Else, the description for all the error codes are returned.

```
EQ,error_code,error_definition_string
```

(<seq_flag='0000'> for the last sequence sent.)

QI - Query Destination Status on a Specific Level by Index

Query the destination status on a specific level using the destination index. This query returns the source that has been taken to the destination on the specified level. Both the destination and level are specified using indices.

Command

```
QI, fullqual_dest_index,lvlIndex
```

Response

```
IQ, fullqual_dest_index,lvlIndex,<'N'|'P'>,<'N'|'C'>,
fullqual_src_index, [fullqual_chop-SrcIndex]
```

All indexes (`dstIndex`, `lvlIndex`, `srcIndex`) are zero–based hex numbers. These indexes refer to the logical index for each entity, not to the physical port number. However, the system can be configured so that port numbers are identical to index numbers. There is no disconnect index number, all indexes refer to configured entities. Valid indexes and their association with specific names can be determined using the commands: `QN, ID`; `QN, IS`; `QN, L`; `QN, IA`.

Example

Suppose the index of POSTPROD is 3, and the index of PROFILE1 is 399. Then the corresponding fullqual_dest_index is `03:018F`. Suppose the level index of video is 0.

To query the status of destination PROFILE1 in area POSTPROD, on video level the following command syntax should be used.

```
QI,03:018F,00
```

QJ - Query Destination Status by Index

Query the destination status by index. This query returns the sources that have been taken to the destination along with the levels on which the source is associated. This command is the index equivalent `QD`.

Command

`QJ` can have different command syntaxes.

`QJ`	In this case, destination status is returned for all destinations for which status has changed since the last time status was sent. Destination status is returned for all levels (for the changed as well as unchanged levels), which have status. The `BK,D` command can be sent before this request to force return of all destination status.
`QJ[,fullqual_dest_index]`	In this case the status for the requested destination will be returned
`QJ[,area_bitmap:]`	In this case the destination status is returned for all destinations in the areas specified by the area bitmap for which status has changed since the last time status was sent to the client.

Response

```
JQ, fullqual_dest_index,nbr_sources
[,src_name_entry1,..., src_name_entryn]
```

(<seq_flag='0000'> for the last sequence sent for a particular destination status.)

`nbr_sources` Number of sources on that destination which are being reported in this message sequence. If there are no sources on this destination at any of its levels, `nbr_sources = 0` will be returned.

src_name_entryn is defined as:

```
<'N'|'P'>,<'N'|'C'>,fullqual_src_index,
level_bitmap,[prot_device_name],
[fullqual_chop_src_index]
```

Parameters for this response are in Table 14.

Table 14. QJ Command Response Parameters

Parameter	Meaning
'N' |'P'	Not-protected or Protected
'N' | 'C'	Not-chopping or Chopping
fullqual_src_index	The source (in the area specified by area index:) currently taken to the destination.
level_bitmap	Describes the levels of that destination for which the source is on.
prot_device_name	The device currently holding the protect. If the destination is not protected, or the device name is unknown, the field is left blank.
fullqual_chop_src_index	The index of the source (along with the source area) chopping to this destination

All indexes (`dstIndex`, `lvlIndex`, `srcIndex`) are zero–based hex numbers. These indexes refer to the logical index for each entity, not to the physical port number. However, the system can be configured so that port numbers are identical to index numbers. There is no disconnect index number, all indexes refer to configured entities. Valid indexes and their association with specific names can be determined using the commands: `QN, ID`; `QN, IS`; `QN, L`; `QN, IA`.

If there are no sources currently on some of the levels defined for the destination, no information is reported for those levels.

If the query is made for all the areas or all the destinations in a given area, the response will contain only those destination whose status has changed since the last query. If there are no destination status changes since the last query, the response will be a blank `JQ`. That is, no destinations status will accompany the `JQ`.

Example

To query the status of all the destinations in all the areas, the following command syntax should be used.

```
QJ
```

QN - Query Names

Note TieLine related commands (`T`, `M`, and `Y`) are not implemented for Encore Software Version 1.6.5, which is the current software release at the time of this manual's revision. Check the Encore Release Notes for software versions released after Version 1.6.5 for the announcement of RCL command implementation.

Query the various configuration names in the system. Typically the names configured in the system are queried by the client, at connect time and whenever the name gets changed. The configuration name change is indicated to the client, by setting the flags associated with `BK,F`. Also if the client has subscribed for name changes, the client will be notified asynchronously.

Command

```
QN,parameter[,area_bitmap]
```

One parameter can be specified per `QN` call. The parameters available, and their meaning, are in Table 15.

Table 15. QN Parameters

Parameter	Meaning
S	Source Names
D	Destination Names
L	Level Names
V	Salvo Names
T[a]	Tie line
M[a]	Tie line Name
Y[a]	Tie line Type
IS	Sources Names with source indices
ID	Destinations Names with destination indices
A	Area Names
IA	Area Names with area indices.

[a] TieLine Commands are not implemented as of Encore Software Version 1.6.5.

The optional area bitmap can be specified only for the parameters `S/IS` and `D/ID`. This area bitmap specifies the areas on which the client requests the names from the server. When the area bitmap parameter is not specified the source or destination names will be returned from all the areas that are present in the network. In case of large systems the number of areas may be significant and the client may be operating only on one or couple of areas. The area bitmap will specify the areas from where the names should be retrieved and returned to the client.

Response S

```
NQ,S,nbr_sources[,src_name_entry1,...,src_name_entryn]
```

src_name_entryn is defined as:

```
fullqual_src_name,<'N'|'T'>,level_bitmap
```

<'N'|'T'> Indicates No-TieLine or Tieline related

Response D

```
NQ,D,nbr_destns
[,dest_name_entry1,...,dest_name_entryn]
```

dest_name_entryn is defined as:

```
fullqual_dest_name,<'N'|'T'>,level_bitmap
```

<'N'|'T'> Indicates No-TieLine or Tieline related

Response L

```
NQ,L,nbr_levels[,lvl_name_entry1,...,lvl_name_entryn]
```

lvl_name_entryn is defined as:

```
lvl_name,lvl_number,< 'R' | 'N' >
```

lvl_number A hex ASCII number from 00 to 1f (representing levels 00 to 31 decimal)

'R' | 'N' Indicates that the level is Restricted or Not Restricted with respect to assignments

Response V

```
NQ,V,nbr_salvos[,salvo_name1,...,salvo_namen]
```

Response T

Note TieLine Commands are not implemented as of Encore Software Version 1.6.5.

```
NQ,T,nbr_tieLines[,tieLine_entry1,...,tieLine_entryn]
```

tieLine_entryn is defined as:

```
tieLineName,<'N'|'R'>, fullqual_beginDstName,
upstreamLvlName, fullqual_endSrcName,
downstreamLvlName
```

<'N'|'R'> Indicates Not-reserved or Reserved

Response M

Note TieLine Commands are not implemented as of Encore Software Version 1.6.5.

```
NQ,M,nbr_tieline_entries
[,tieline_entry1,.., tieline_entryn]
```

`tieline_entry` is defined as:

```
tieline_Name,<'N'|'R'>,tlTypeName,
fullqual_beginDstName, fullqual_endSrcName
```

Response Y

Note TieLine Commands are not implemented as of Encore Software Version 1.6.5.

```
NQ,Y,nbr_tltype_entries[,tltype_entry1,..,
tltype_entryn]
```

`tltype_entryn` is defined as:

```
tltype_name,lvlbitmap
```

`lvlbitmap` includes the EndLevels in this entry, and for each consecutive 1 in the EndLevels bitmap (least significant bit first), a `lvlbitmap` of the BeginLevels that feed this endLevel. If there are many EndLevels in the tieline type, it may take more than one `tltype_entry` to send them all.

Response IS

```
NQ,IS,nbr_sources
[,IS_src_name_entry1,...,IS_src_name_entryn]
```

`IS_src_name_entryn` is defined as:

```
fullqual_src_name,fullqual_src_index,<'N'|'T'>,
level_bitmap
```

`<'N'|'T'>` No-tieline or Tieline Related

Response ID

```
NQ,ID,nbr_destns
[,ID_dest_name_entry1,..., ID_dest_name_entryn]
```

`ID_dest_name_entryn` is defined as:

```
fullqual_dest_name,fullqual_dest_index,<'N'|'T'>,
level_bitmap
```

`<'N'|'T'>` No-tieline or Tieline Related

Response A

```
NQ,A,nbr_areas[,area_name1,...,area_namen]
```

Response IA

```
NQ,IA,nbr_areas
[,area_name_entry1,...,area_name_entryn]
```

`area_name_entryn` is defined as:

```
area_name, area_index
```

For all responses <seq_flag='0000'> for the last sequence sent.

Example

Suppose a client is interested in areas NEWSROOM and INGEST. The area index of NEWSROOM is 22 and that of INGEST is 2.

To query the source names (with indices) in the above areas the client needs to issue following command:

```
QN,IS,0000000000400004
```

To query the destinations in all the areas the client can issue the command:

```
QN,D
```

QV - Query Salvo Elements

Query the salvo elements which make up the salvo. The salvo is specified by name.

Command

```
QV,salvo_name
```

Response

```
VQ,salvo_name,nbr_entries[,entry1,...,entryn]
```

(<seq_flag='0000'> for the last sequence sent.)

`entryn` is defined as follows:

```
fullqual_dest_name, fullqual_src_name,level_bitmap
```

Example

To query the salvo elements that comprise a salvo named `RCLSVO1` the client needs to send

```
QV,RCLSVO1
```

RC - RCL Connect

Request RCL connect to the RCL server.

Command

`RC`

The RCL server on getting a `RC` request from a client will check the validity of the client and accept the client request. A unique session_id is generated for the connection. Then the server responds back intimating the connection acceptance by sending the following response.

Response

`RA, session_id`

`session_id` The unique session_id created for this connection by the server. 2 Hex digits with values from 0-0xFF.

If the client reissues a `RC`, the session will be reinitialized. All the subscriptions made so far and the state information will be lost.If the client is not configured in the user interface, the connection will be closed without any response.

RD - RCL Disconnect

Request RCL disconnect.

Command

`RD`

Response

This will get an `ER,00` response on success and an `ER,nn` (nn>0) on failure indicating the reason of failure.

This specifies the RCL disconnect request for the current session. Either the client or the server can send this. The end that receives disconnect request will respond back with an `ER, 00` and terminate the session. The client will send this when it wants to close the session. For graceful disconnect the client should send `RD` before closing the socket or serial port.

TA – Take (Breakaway)

Take the sources on the specified levels to the destination by name. `TA` facilitates performing breakaway takes in a single request by name. The client can specify individual sources along with the levelbitmap information to take different sources on different levels. VITC time stamp can also be provided to perform deterministic switching.

Command

```
TA,time_stamp,fullqual_dest_name,nbr_sources,
src_name_entry1[,..., src_name_entryn]
```

(<seq_flag='0000'> for the last sequence sent.)

`time_stamp`	Time at which the take will be performed.
`fullqual_dest_name`	Destination to be taken to.
`nbr_sources`	Number of following entries (must be at least one).

`src_name_entry` is defined as:

```
fullqual_src_name,level_bitmap
```

Response

This will get an `ER,00` response on success and an `ER,nn` (nn>0) on failure indicating the reason of failure.

Example

The example shows how to perform a breakaway take of the following sources on the levels specified to destination MON1 in area MIXROOM.

Source - MAINCAM Level index – 3

Source - ANCCAM Level index - 2

Source – FEED Level index – 1

All the above sources are in area MIXROOM.

```
TA,, MIXROOM:MON1,3, MIXROOM:MAINCAM,00000008,
MIXROOM:ANCCAM,00000004, MIXROOM:FEED,00000002
```

The above take was non time stamped and thus the timestamp field is blank. However the separator needs to be sent as part of the message.

Note Separators are horizontal tabs ([HT]) represented by the comma (,) character. In the example, the comma after the TA, represents the horizontal tab for the blank time_stamp field. For more information on notation see *Command and Message Description Notation* on page 12.

TD - Take (Single Source)

Take the source on the specified levels to the destination by name. The level information can be omitted from this command thus allowing to perform an All Level take. VITC time stamp can also be provided to perform deterministic switching.

Command

```
TD,time_stamp, fullqual_dest_name,src_name_entry
```

`time_stamp` Time at which the take will be performed.
`fullqual_dest_name` Destination to be taken to

`src_name_entry` is defined as:

```
fullqual_src_name[,levelbitmap]
```

With no level bitmap specified, the source will be taken to all levels of the destination.

Response

This will get an `ER,00` response on success and an `ER,nn` (nn>0) on failure indicating the reason of failure.

Example

To take the sources `FEED1` in area `INGEST` on the level indices 2,3and 4 to the destination `VW3` in area `POSTPROD` the following syntax is used:

```
TD,,POSTPROD:VW3,INGEST:FEED1,0000001C
```

The above take will need a tie-line to have been defined in the system between areas `INGEST` and `POSTPROD`.

TI - Take by Level Index

Take the source on the specified levels to the destination by index. The level information can be omitted from this command thus allowing to perform an all level Take. VITC time stamp can also be provided to perform deterministic switching. In this command the source, destination and level should be specified using their indexes.

Command

```
TI,time_stamp, fullqual_destIndex, fullqual_srcIndex
[,levelIndex]
```

`time_stamp` Time at which the take will be performed.

All indexes (`dstIndex`, `lvlIndex`, `srcIndex`) are zero–based hex. These indexes refer to the logical index for each entity, not to the physical port number. However, the system can be configured so that port numbers are identical to index numbers. There is no disconnect index number, all indexes refer to configured entities. Valid indexes and their association with specific names can be determined by using the commands: `QN,L`; `QN,ID`; `QN,IS`.

Response

This will get an `ER,00` response on success and an `ER,nn` (nn>0) on failure indicating the reason of failure.

Example

To take the source FEED1 with index 3 in area INGEST whose index is 5 on all levels to the destination VW3 (index 1) in the same area the following syntax is used:

```
TI,,05:0001,05:0003
```

TJ - Take by Level Bitmap

Take the sources on the specified levels to the destination by index. `TJ` facilitates performing breakaway takes in a single request by index. The client can specify individual sources along with the levelbitmap information to take different sources on different levels. VITC time stamp can also be provided to perform deterministic switching.

Command

```
TJ,time_stamp, fullqual_dest_index,nbr_sources,
src_name_entry1[,..., src_name_entryn]
```

(<seq_flag='0000'> for the last sequence sent.)

`time_stamp`	Time at which the take will be performed.
`fullqual_dest_index`	Destination to be taken to
`nbr_sources`	Number of following entries (must be at least one)

`src_name_entryn` is defined as:

```
fullqual_src_index,level_bitmap
```

All indexes (`dstIndex`, `lvlIndex`, `srcIndex`) are zero–based hex numbers. These indexes refer to the logical index for each entity, not to the physical port number. However, the system can be configured so that port numbers are identical to index numbers. There is no disconnect index

number. All indexes refer to configured entities. Valid indexes and their association with specific names can be determined by using the commands: `QN,L`; `QN,ID`; `QN,IS`.

Response

This will get an `ER,00` response on success and an `ER,nn` (nn>0) on failure indicating the reason of failure.

Example

To perform a timed take at 16 hours, 23 minutes, 10 seconds and 15 th frame of the source FEED1(index 3) in area INGEST (index 5) on the levels 0 and 1 to the destination VW3 (index 1) in the same area the following syntax is used:

```
TJ,16231015,1,05:0001,05:0003,00000003
```

TM - Take Monitor

Note The `TM` command is not implemented for Encore Software Version 1.6.5, which is the current software release at the time of this manual's revision. Check the Encore Release Notes for software versions released after Version 1.6.5 for the announcement of RCL command implementation.

Take the source or destination as specified by the `mon_type` to the monitor names `mon_name`.

Command

```
TM,Mon_name,Mon_type,type_name
```

`Mon_name`	Configured monitor name | Default monitor name
`Mon_type`	'S' = Source, 'D' = Destination
`type_name`	Source or Destination Name

Response

This will get an `ER,00` response on success and an `ER,nn` (nn>0) on failure indicating the reason of failure.

TS - Take Salvo

Execute the salvo specified by the `salvo_name`. VITC time stamp can also be provided to perform deterministic switching.

Command

`TS,time_stamp,salvo_name`

`time_stamp` Time at which the take will be performed.

`salvo_name` Salvo name

Response

This will get an `ER,00` response on success and an `ER,nn` (nn>0) on failure indicating the reason of failure.

Example

To perform a Salvo take of a salvo name `RCLSVO1` the following command syntax should be used:

`TS,,RCLSVO1`

UP - Request Unprotect

Note Only an All Level Unprotect is implemented for Encore Software Version 1.6.5, which is the current software release at the time of this manual's revision. Check the Encore Release Notes for software versions released after Version 1.6.5 for the announcement of RCL command implementation.

Unprotect command allows the protect on the specified destination to be removed on the specified level. The command requires a `level_bitmap` to be specified on which unprotect should be performed. The format of the Unprotect command is shown below.

Command

`UP,fullqual_dest_name,level_bitmap`

Response

This will get an `ER,00` response on success and an `ER,nn` (nn>0) on failure indicating the reason of failure.

Example

To Unprotect a destination named `PDR2` in area `POSTPRD` on levels with indices 3 and 4 the client should issue the following command:

`UP,POSTPRD:PDR2,00000018`

Client Subscription Message Descriptions

SB – Subscribe

Subscribe request for status/configuration change information. The specific information for which the client subscribes is decided based on the subscription type and parameters sent as part of the `SB` command.

Command

```
SB[, subscription_type[,parameters]]
```

The subscription type specifies the status or particular configuration element. Refer to Table 16 for the list of available subscription type acronyms. If no subscription type is specified in the `SB` command, then a global subscription for all supported events is done. The parameters for any subscription command depend on the subscription type and are explained in the *Subscription Commands Usage* on page 55.

Response

Subscription request will get an `ER,00` response on success and an `ER,nn` (nn>0) on failure indicating the reason of failure.

UB – Unsubscribe Request

Unsubscribe request for already subscribed status/configuration change information. The specific information for which the client un-subscribes is decided based on the subscription type and parameters sent as part of the `UB` command.

Command

```
UB[, subscription_type[,parameters]]
```

The subscription type specifies the status or particular configuration element. Refer to Table 16 for the list of available subscription type acronyms. If no subscription type is specified in the `UB` command, then a global unsubscription for all subscribed events is done. The parameters for any unsubscribe command depend on the subscription type that needs to be unsubscribed and are explained in the *Subscription Commands Usage* on page 55.

Response

Unsubscribe request will get an `ER,00` response on success and an `ER,nn` (nn>0) on failure indicating the reason of failure.

Server Originated Message Descriptions

NY – Notification

An asynchronous Notification is sent to the client by the server whenever the subscribed status/configuration information changes.

Notification Format:

```
NY, subscription_type[,parameters]
```

Refer to Table 16 for the list of available subscription type acronyms. The parameters for the notification depend on the subscription type and are explained in the *Subscription Commands Usage* on page 55.

Table 16. Subscription Types

Subscription_type	Meaning
DS	Destination status change
CN	Source or destination configuration change
LV	Level configuration change
SL	Salvo configuration change
AR	Area configuration change
SV	Salvo content change

RD - RCL Disconnect

Request RCL disconnect.

Command

```
RD
```

Response

This will get an `ER,00` response on success and an `ER,nn` (nn>0) on failure indicating the reason of failure.

This specifies the RCL disconnect request for the current session. Either the client or the server can send this. The end that receives disconnect request will respond back with an `ER, 00` and terminate the session. The client will send this when it wants to close the session. For graceful disconnect the client should send `RD` before closing the socket or serial port.

RN - RCL Announce

The RCL server during start up will send a command called RCL Announce (RN) to all the connected clients indicating that the server has just started up.

Command

```
RN
```

Response

None

Subscription Commands Usage

This section describes the usage of the subscribe (SB),unsubscribe (UB) and the notification (NY) commands. These commands are described with respect to the following types of subscription

- *Status Change Subscriptions* on page 55
- *Configuration Change Subscriptions* on page 57

Status Change Subscriptions

A RCL client can subscribe for destination status of one or a group of destinations. Whenever the status of a destination changes, all the clients that have subscribed for that destination status receive a notification.

Subscribe for Status Change

Subscribe command for destination status could be used in the following formats.

Command

```
SB, DS
```

The client receives notifications, whenever the status of any destination in any area changes.

Command

```
SB, DS, area index
```

The client receives notifications, whenever the status of any destination in the area specified by the area index changes.

Example

To subscribe for destination status changes in area POSTPRD (index 2) the client should issue the following command: `SB,DS,02`.

```
SB, DS, fullqual_dest_name
```

The client receives notification only if the status of the specified destination changes.

Unsubscribe for Status Change

Unsubscribe command for destination status could be used in the following formats.

Command

```
UB, DS
```

Unsubscribe destination status change notifications for all destinations in all areas

Command

```
UB, DS, area index
```

Unsubscribe destination status change notifications in the area specified by the area index

Command

```
UB, DS, fullqual_dest_name
```

Unsubscribe the specified destination change notification.

Example

To unsubscribe for destination status changes for destination `VW3` in area `POSTPRD`, the client should issue the following command:
`UB,DS,POSTPRD:VW3`

Illegal Subscribe and Unsubscribe Combinations

Do not use these subscribe and unsubscribe combinations:

```
SB,DS and UB,DS, fullqual_dest_name
or
SB,DS, area bitmap and UB,DS, fullqual_dest_name
```

In these cases an appropriate error will be sent back to the client.

Notification for Status Change

Whenever the status for a destination changes, all RCL clients that have subscribed for that destination's status receive notifications in the following format.

```
NY, DS, fullqual_dest_name, nbr_sources
[,src_name_entry1,..., src_name_entryn]
```

`src_name_entryN` is defined as

```
<'N'|'P'>,<'N'|'C'>, fullqual_src_name,level_bitmap,
[prot_device_name], [fullqual_chopsrc_name]
```

Note The notification format is similar to the `DQ` response.

Configuration Change Subscriptions

A RCL client can subscribe for changes in the configuration of the system. The configuration changes that can be subscribed for are described along with their command format in this section.

Subscribe for Configuration Change

Subscribe command for Configuration change could be used in the following formats.

Command

```
SB, CN [, area index]
```

Subscribe to get notified, when destination or source configuration changes in the router. When the optional area index is specified, the server will send notification for changes only in the specified area.

Command

```
SB, LV
```

Subscribe to get notified, when level configuration changes.

Command

```
SB, SL
```

Subscribe for salvo configuration changes.

Command

```
SB, AR
```

Subscribe for area configuration changes.

Command

```
SB, SV
```

Subscribe for content changes of any salvo.

Example

To Subscribe for source and destination configuration changes in areas POSTPRD (index 4) and EDITROOM (index 2), the client should issue the following commands:

```
SB,CN,04
SB,CN,02
```

Unsubscribe for Configuration Change

The previously subscribed configuration change subscriptions can be unsubscribed using the Unsubscribe command syntaxes shown below.

Command

```
UB, CN [, area index]
```

Unsubscribe the source and destination configuration change subscription. The client can choose to unsubscribe only in specific area by specifying the area index.

Command

```
UB, LV
```

Unsubscribe level configuration change subscription.

Command

```
UB, SL
```

Unsubscribe salvo configuration change subscription.

Command

```
UB, AR
```

Unsubscribe area configuration change subscription.

Command

```
SB, SV
```

Unsubscribe the previously subscribed salvo content change subscription.

Notification for Configuration Change

Whenever a configuration change occurs in the system, all the RCL clients that have subscribed for that configuration change receive a notification from the RCL server with the subscription type specifying the element which has changed. In case of source and destination name changes, the clients also receive additional area bitmaps specifying the areas in which the names have changed.

```
NY, subscription_type [, area index]
```

`subscription_type`	One of the subscription types specified in Table 16 (Except `DS` and `SV`).
`Area index`	Specified only if the subscription type is `CN`. The area index indicates the area on which the destination or source configuration has changed.

Upon receiving this notification, the client has to invoke specific commands to refresh the changed configuration.

Example

If the client receives a notification with the subscription_type as `SL`, then the client should issue a `QN, V` to reload the fresh salvo names from the system.

In case of salvo content change subscriptions, whenever the content of a salvo changes, the following notification is sent to all clients that have subscribed for that change.

Command

```
NY, SV, salvo name
```

When a client receives such a notification, the client has to send a salvo status query (`QV`) for the notified salvos to refresh their contents.

Section 3

Series 7000 Native Protocol

Introduction

The Series 7000 Routing System can be controlled by an external, serially communicating device such as a personal computer or an automation system. The Native Protocol described is intended to facilitate computer control of the Series 7000. A dumb terminal is not a practical Series 7000 controller.

Commands and error responses are terse and character efficient to maximize throughput. All message bytes are from the ASCII character set (printable and control characters). This provides the ability to easily log information traveling through the duplex control ports.

Command and Message Description Notation

For the command parameter descriptions in this section, lower case parameters must be replaced by user specified information, while upper case parameters must be literally supplied.

Upper case parameters fall into two categories: printable ASCII characters, where they are supplied as shown, and ASCII control characters, where the text shown translates into a hex equivalent.

Notation symbols used in the format descriptions are shown in Table 17.

Table 17. Notation symbols

Symbol	Meaning
...	A continued sequence.
|	Or
[]	Optional parameters
< >	Choices, or ASCII control characters, or for clarity.
,	Comma designates horizontal tab <HT>, the data separator.

For the sake of readability, spaces may be shown in the descriptions where none exist in the protocol definition.

Interface Requirements

In order to control a Series 7000 system using Native Protocol, both the Series 7000 and the external device must be properly equipped.

Communication with the Series 7000 system is via either an RS-232 or RS-422 interface, or via an Ethernet interface.

RS-232 or 422 Communication

Series 7000 systems with a stand alone control frame must be equipped with a Communications Interface (CIF) module and an RS-232/422 Interface board.

Series 7000 compact frames require an Asynchronous Mezzanine (Amezi) in the upper mezzanine position. The use of Native Protocol via a SLIP connection to a compact router is not recommended or supported. While it is possible, the slow SLIP communications link is likely to cause problems with the client application. Performance would be degraded even further if either the GUI or VSD were operated concurrently. These programs would probably fail because of communications timeouts.

Default Series 7000 communication settings are:

- RS-232 Protocol
- 9600 Baud
- 8 Data Bits
- 1 Stop Bit
- No Parity

The external device controlling the Series 7000 must be equipped with an RS-232 or RS-422 Serial Port capable of 300, 600, 1.2k, 2.4k, 9.6k, 19.2k, or 38.4k Baud communication rates.

Pinouts and cable diagrams for creating RS-232 or RS-422 connections are available as part of the Installation instructions in Grass Valley Series 7000 product manuals.

Ethernet Communication

Each Series 7000 stand alone control frame is equipped with an Ethernet port. Use of 10base2 Ethernet, 50 Ohm coax, tees, and terminations is recommended. The network should be closed, for use only by the Series 7000 system. Configurations that are connected to larger, open networks, are not supported.

Each device on the network must be assigned a unique IP address and name. This includes each Series 7000 MCPU, both primary and backup, plus each PC or other computer. Refer to the Grass Valley Series 7000 product manuals for a discussion on interconnection and addressing using Ethernet. Also review the instructions for Ethernet interface hardware in your computer manuals. It may be necessary to consult an expert in the field of Ethernet network installation.

To use the Native Protocol Ethernet interface, user-supplied software must be created to send and receive Native Protocol messages according to the protocol specifications in this document. The programmer writing software for this application must be skilled in the use of serial communications protocols, and the use of TCP/IP stream sockets. Knowledge of Ethernet networking and system administration is also required to install and configure software.

Series 7000 compact systems do not support Ethernet. External control using Native Protocol control is accomplished via an RS-232 or RS-422 interface.

Basic Native Protocol Description

The levels of the Series 7000 Native Protocol are defined in Table 18:

Table 18. Protocol Levels

Level	Description
Level 1	Physical (e.g. RS-232, RS-422, Ethernet)
Level 2	Data Link (e.g. checksums, ACK/NAK)
Level 3	Supervisory (e.g. flow control, message buffering)
Level 4	Application

The following discussions assume that Levels 1, 2, 3, and 4 will be similarly implemented on each end of the communication link.

Because of significant differences in all but level 4 messages, the RS-232/RS-422 and Ethernet descriptions are presented separately.

RS-232 and RS-422 Description

Level 1

- RS-232 or RS-422 (Default = RS-232).
- Baud rate - 300, 600, 1.2k, 2.4k, 9.6k, 19.2k, or 38.4k (Default = 9.6k)

Level 2

- 1 stop bit
- 8 data bits
- No parity

Level 2 adds the <SOH> character and the protocol_id to the message byte stream. It calculates the message checksum and appends it to the message. It then adds the <EOT> character, and transmits the message. Level 2 double buffers output.

The receiving end buffers input, verifies the <SOH>, protocol_id, and <EOT> bytes, and verifies the checksum. If the message is successfully received, it notifies Level 3 of its availability.

Level 2 ACK/NAK

When a message is successfully received, an ACK (0x06) message is returned to the sender. ACKs are returned immediately, with no field delay. ACKs are not encapsulated in <SOH>,<checksum>, or <EOT>. If the sender does not receive an ACK within a reasonable time interval, the message is re-transmitted for a total of thirty (30) attempts. The 7000 end of the protocol will always attempt to transmit exactly thirty (30) times. However, the external device may choose fewer attempts, or simply keep transmitting the same message until it is finally acknowledged.

If an error occurs during reception of a message, a NAK (0x15) followed by an error code descriptor is returned to the sender. NAKs are returned immediately, with no field delay. NAKs and Level 2 error codes are not preceded by an <SOH> or verified with a <checksum>. However an <EOT> trails the NAK message, as follows:

```
<NAK> <ErrorCodeHI> <ErrorCodeLO> <EOT>
```

The error code is a two digit hex number, expressed in ASCII.

The sender can attempt to re-transmit. However, the sender should not attempt transmission of a new or repeated message until an ACK or NAK is received, or until an appropriate time-out occurs. If a NAK with an error code indicating buffer not available is received, the sender should delay before attempting a re-transmission.

When the external device returns NAKs to the Series 7000, they must be in the format described above (with error code and EOT).

See *Level 2 (NAK) Error Code Descriptions* on page 77 for specific error information.

Level 3

Level 3 copies input messages from Level 2 buffers into its own buffers (up to "n" buffers). By marking the Level 2 buffers as available, it effectively accomplishes flow control (assuming the sender delays long enough before attempting to re-transmit the message, and the receiver gets a buffer cleared out in time). Should buffers become full, Level 2 will return the required <NAK> <buffer not available error> <EOT>, for every message attempted while this buffer (or queue) full condition remains. Again, a delay between transmitted messages should be invoked when this error condition is reported to the sender.

Level 3 passes incoming data buffers up to the appropriate Level 4 protocol handler one at a time. Level 3 appends the Series 7000 internal header to the front of the Native Protocol message before delivering to Level 4.

Level 3 receives output messages from Level 4, one at a time, and buffers them for output to Level 2. Level 3 strips the Series 7000 internal header from the message, only passing on to Level 2 the properly formatted Native Protocol message. Messages are passed to Level 2 one at a time. Level 2 calculates the checksum and transmits.

Level 3 Error Recovery

Thirty (30) consecutive transmit retries of a message will be attempted. Should the packet not be accepted by the receiving side within that count, an error will be sent by the sending side to the MCPU resident Redundancy Task for appropriate action. Appropriate action could be a switch over to the redundant Interface Card if redundant pairs are involved, or an Interface reset in stand-alone configurations.

In both transmit and receive cases, the retry count is reset to zero on any respective ACK.

See *Error Codes* on page 77 for specific error information.

Level 4

Level 4 of Native Protocol parses the content of the messages, and accesses the Series 7000 to either return the information requested, or to perform the requested action.

See *Native Protocol Messages* on page 81 for a detailed listing of all Level 4 messages.

Ethernet Description

Level 1 Description

- 10 Base 2 Ethernet
- Closed network strongly recommended (dedicated to Series 7000)

Levels 2, 3, and 4 Description

Stream sockets use TCP (Transmission Control Protocol), which is a stream oriented protocol. These sockets (sometimes referred to as Berkeley sockets) treat communications as a continuous stream of characters and are connection oriented. Therefore, a connection must be opened and maintained for the duration of the communications. Stream sockets are supported for many different host environments and operating systems.

To Initiate Communications:

A computer or other host device using Native Protocol must originate communications with the MCPU. The computer is a client, and the MCPU is the server.

1. Create a stream socket on the client.

   Linger options should be set as required for the application. Linger on with a timeout of zero is a good starting point.

2. Connect the socket to the IP address of the desired MCPU, on port 12345.

   When the connect succeeds, the MCPU will report an `NPi device added` on the system diagnostic console (if device event logging is enabled).

Note If there are redundant MCPUs, each MCPU must have its own unique IP address and name. However, only the active (primary) MCPU can communicate; it does not connect to the backup MCPU. If an MCPU switchover occurs, communications will be lost with the previously active MCPU, and it will be necessary to reconnect with the newly active MCPU. One approach is to create two sockets and attempt to connect to both MCPUs; the active (primary) connections will succeed, and the backup will not.

3. End the connection from the client by closing the socket.

   This will be detected by the MCPU, which will report the `NPi device deleted` on the system diagnostic console (if device event logging is enabled).

The user-supplied software should be designed to be able to recognize the SIGPIPE (broken pipe) interrupt or its equivalent. This will allow detection if the connection is terminated by the MCPU. Otherwise, systems may not detect termination until a message is sent.

At the MCPU system diagnostic console, Native Protocol devices which are connected via Ethernet are referred to as NPi devices (Native Protocol/Internet). System diagnostic commands in Table 19 may be used:

Table 19. System Diagnostic Commands

Command	Meaning
ls anpi	lists Active NPi devices
pr cnpi "NPI1"	displays NPI configuration
ls inet	lists active NPi and RNC devices

By default, an NPi device name is constructed for listing purposes which consists of the last two IP address digits. For NPi devices to have names, the HOSTS table in the MCPU flash file system must contain the correct name/IP address relationships. (Refer to the configuration instructions in Grass Valley product manuals for more information on the HOSTS file).

Data Link and Supervisory Control

Unlike Native Protocol via Amezi, the TCP/IP communications layers used with Ethernet are responsible for the end-to-end error-free transport of messages. This means that messages sent and received via stream sockets are guaranteed to arrive in order and error-free, as long as the connection between the client and server is maintained.

In Native Protocol via an Amezi, Levels 2 and 3 are responsible for the error-free transport of messages. Since data transport is managed transparently for TCP/IP stream sockets, the `ACK/NAK` protocol is not used for Ethernet communications. The user-supplied software must not generate or expect `ACK`s or `NAK`s for message transactions. Level 2 error messages will not be generated.

Since the TCP stream connection is error-free, message format and checksum errors are generally a result of a programming error and not a communications error. The user-supplied program should be able to prevent these errors.

Sending Messages

While connected, Native Protocol Level 4 messages may be sent from the client by writing to the stream socket. Each message sent must be properly constructed as documented for Native Protocol messages; each message must begin with a `SOH`, end with an `EOT`, and the transmitted checksum must be correct. The user-supplied software must check for errors returned by the socket's write function call to ensure that the entire message was accepted and transmitted correctly.

All Native Protocol Level 4 messages and responses are available to an Ethernet client. However, since the Level 2 ACK/NAK is not used the BK,2 command will be ignored (no response is generated).

If a message is not sent for a period of time which exceeds the timeout configured or set for this connection, then the server (MCPU) will close the connection, and communications will be terminated. If a nonzero timeout is set, the user-supplied software must ensure that a message (any message) is sent periodically. The timeout value may be set by the user-supplied software program by transmitting a BK,I command. If the timeout is not set with a BK,I command, then the default value configured in the **CFGD NATIVE PROTOCOL INET** dialog box **REFRESH RATE** in the GUI will be used.

The rate at which messages are sent to the MCPU should be regulated to prevent overrunning the Native Protocol processing task in the MCPU application. If too many messages are sent too quickly the message buffers in the MCPU will fill, which will block the call to the socket's write function. To avoid this, wait until a Level 4 response is received from a previous command message before sending another. This ensures that the MCPU processing task has time to service all devices. Turn on the command echo option, either as the default in the **CFGD NATIVE PROTOCOL INET** dialog box in the GUI, or by using the BK,E,ON command message.

One possible source of problems is in the checksum verification. Message checksums as defined for Level 2 must be calculated and included with all messages. The MCPU server interface will discard any message with a checksum error, and a Level 2 error message will not be returned. Since TCP/IP guarantees an error-free message, the checksum cannot be corrupted during transmission. However, if the user-supplied software incorrectly calculates the checksum, the message will not be processed by the MCPU.

With stream sockets, it is the responsibility of the user-supplied software to correctly handle byte ordering and padding for multi-byte values. However, since all Native Protocol messages comprise single byte values (ASCII characters), byte ordering is not a problem as long as the correct message format is followed.

Receiving Messages

Native Protocol Level 4 responses are received by the client by reading from the connected stream socket. Each message is formatted as documented, beginning with a SOH character and ending with an EOT character.

When reading a TCP/IP stream socket, data is presented error-free and in the order sent. However, there is no built-in method for identifying the boundaries of messages; it is up to the user-supplied software to look for the beginning SOH and ending EOT. Be aware that most stream socket implementations may deliver message fragments when the read function call is made. The user-supplied software must be designed to buffer received messages until a complete message is received.

Response messages will be received for all command messages sent which specify that return information. A Level 4 acknowledgment response may optionally be returned for messages which do not automatically return a response. Refer to the **CmdEcho** option in the **Cfgd Native Protocol Inet** dialog box in the GUI, or the `BK,E,ON` command message.

Note The default CmdEcho option setting in the GUI applies to all Ethernet Native Protocol devices, but the `BK,E` command applies to each device independently.

Message Formats

Commands received by the 7000 through any control port configured for Native Protocol are formatted as follows.

Request Command Message Format

```
<SOH> <protocol_id> <seq_flag> <request_cmd>
[,parameter(s)] <checksum> <EOT>
```

`<SOH>`	ASCII Start of Header character (0x01)
`<protocol_id>`	One printable ASCII character. 'N' identifies the Native Protocol.
`<seq_flag>`	ASCII '0' if this is the last (or only) message in this sequence. Any other ASCII character indicates that there are more messages coming to complete the data portion of the <request_cmd>.
`<request_cmd>`	Two printable ASCII characters.
`[,parameter(s)]`	Optional parameters (printable ASCII characters) (max 108 bytes). The parameter delimiter is <HT> Horizontal Tab (0x09), and precedes each parameter, including the first. (Note that in the examples, A commas is used to signify <HT>.) A trailing <HT> following the last parameter is optional for messages sent to the 7000 by the external device.
`<checksum>`	Negative sum mod 256 of all previous byte values (not including <SOH>). The one byte checksum is broken into two hex digits, converted to ASCII representation of those two digits, and sent as two ASCII characters. The most significant hex digit is sent first.
`<EOT>`	ASCII End of Transmission (0x04)

Response Command Message Format

For RS-232/RS-422 interfaces, all received commands are acknowledged with an `ACK` or `NAK` by Level 2. These do not exist with the Ethernet internace. In both interfaces some commands have a Level 4 response, described below.

```
<SOH> <protocol_id> <seq_flag> <response_cmd> <,data>
<checksum> <EOT>
```

`<SOH>`	ASCII Start of Header character (0x01)
`<protocol_id>`	One printable ASCII character. “N” identifies Native Protocol.
`<seq_flag>`	ASCII ‘0’ if this is the last (or only) message in this sequence. Any other ASCII character indicates that there are more messages coming to complete the data portion of the <response_cmd>.
`<response_cmd>`	Two printable ASCII characters, different from the “request_cmd”. This difference identifies bus directionality, resolving any ambiguities about the meaning of the data (or parameters) within the command.
`<,data>`	The requested information in printable ASCII (max 108 bytes). The datum delimiter is <HT> Horizontal Tab (0x09), and precedes each datum, including the first. (Examples show <HT> as comma.) A trailing <HT> following the last datum is always sent to facilitate message parsing by the external device.
`<checksum>`	Negative sum mod 256 of all previous byte values (not including <SOH>). The one byte checksum is broken into two hex digits, converted to ASCII representation, and sent as two ASCII characters. The most significant hex digit is sent first.
`<EOT>`	ASCII End of Transmission (0x04)

Level 4 responses occur as soon as the incoming command is processed, with no field delay.

Level 4 Response Message (ACK Level 4) Format

Some applications may require that control operations be successfully completed before another command is sent. This requires an acknowledgment from Level 4. This acknowledgment assures the external controller that the command has been sent on to the MCPU, but not that the MCPU has completed execution of the command. The acknowledgment will be one of three possible responses:

- Expected response containing data.
- An error response. See *Error Codes* on page 77 for specific information.
- If neither of the above occurs, Level 4 will return an error response with error number = ØØ (i.e., "no error occurred.")

```
<SOH> <protocol_id> <seq_flag> <ER,00> <,request_cmd,>
<checksum> <EOT>
```

`<SOH>`	ASCII Start of Header character (0x01)
`<protocol_id>`	One printable ASCII character. "N" identifies Native Protocol.
`<seq_flag>`	ASCII '0' if this is the last (or only) message in this sequence. Any other ASCII character indicates that there are more messages coming to complete the data portion of the <response_cmd>.
`<ER,00>`	ASCII Error 00
`<,request_cmd>`	The two-letter designation of the offending command. If the error response originates from the RS-232/422 Interface card, this will be exactly the command sent by the external device. However, if the error response originates at the MCPU, there is no way to correlate the response with the requesting command — in this case the returned <request_cmd> = MC.
`<checksum>`	Negative sum mod 256 of all previous byte values (not including <SOH>). The one byte checksum is broken into two hex digits, converted to ASCII representation, and sent as two ASCII characters. The most significant hex digit is sent first.
`<EOT>`	ASCII End of Transmission (0x04)

These responses occur as soon as the incoming command is processed, with no field delay. Level 4 error response can be enabled or disabled from the Graphic User Interface, or by using the `KB,E` command.

Level 4 Error Message Format

The format for error messages originating at Level 4 returned to controlling devices through any control port configured for Native Protocol is as follows.

```
<SOH> <protocol_id> <seq_flag='0'> <ER> <,error_code>
<,request_cmd> [,data] <checksum> <EOT>
```

`<SOH>`	ASCII Start of Header character (0x01)
`<protocol_id>`	One printable ASCII character. "N" identifies Native Protocol.
`<ER>`	The two ASCII characters 'ER'
`<,error_code>`	Unique two digit ASCII code defining the error detected at Level 4. Level 4 error codes are two digit hex numbers, and are transmitted as two ASCII bytes, most significant first. Leading 0's are sent. "ER,00" is reserved for Level 4 Acknowledgment Format, and is not really an error.
`<,request_cmd>`	The two-letter designation of the offending command. If the error response originates from the RS-232/422 Interface card, this will be exactly the command sent by the external device. However, if the error response originates at the MCPU, there is no way to correlate the response with the requesting command — in this case the returned <request_cmd> = MC.
`[,data]`	Optional printable ASCII information providing additional information about the error. In many cases, this data will be one of the parameters of the failed command (e.g., the incorrect dest_name). The datum delimiter is <HT> Horizontal Tab (0x09), and precedes each datum, including the first. (Examples show <HT> as comma.) A trailing <HT> following the last datum is sent to facilitate parsing by the external device. Incoming control messages are buffered to maximize throughput. If Level 4 Acknowledgment is not active, error messages may not be synchronous with control messages, i.e., an error message may relate to a command sent several commands back. By providing the name of the offending command, along with the offending parameter, the error should be identifiable.

`<checksum>`	Negative sum mod 256 of all previous byte values (not including <SOH>). The one byte checksum is broken into two hex digits, converted to ASCII representation of those two digits, and sent as two ASCII characters. The leftmost hex digit is sent first.
`<EOT>`	ASCII End of Transmission (0x04)

These responses occur as soon as the incoming command is processed, with no field delay.

Message Sizes and Sequences

Message sizes can grow quite large, both for commands and responses. For example, the `TA` (Request Take) command could grow large, depending on the number of sources and levels specified. Long commands and responses may need to be segmented and sent in a sequence of messages rather than in one large message. For this reason, `seq_flag` was added to the Level 4 protocol.

- If a message is received with <`seq_flag` = any ASCII printable character not equal to '0'>, there are more segments to come.
- If <`seq_flag` = ASCII '0'>, this is the last (or only) segment.

Some messages are not candidates for sequencing. These messages have their sequence flags set = to ASCII '0' (<`seq_flag` = '0'>). Messages that are sequencing candidates are not necessarily sent sequenced. Message sequencing occurs only if the size of the message exceeds the buffer size specified by the Native Protocol (see *Message Buffer Sizes* below). For example, the `TA` command specifies the `nbr_sources` in the data portion of the message. If the total number of sources = 1, only one sequence will be sent. However, if the total number of sources = 10 it may be necessary to split up the total message into several messages. Each of the sequenced messages has `nbr_sources` set such that the byte count is smaller than the buffer size. For example, four sequences may have to be sent in order to describe all 10 sources, with `nbr_sources` = 4, 2, 3, and 1.

Messages are sent with entries intact, so each message makes complete sense on its own. Messages are not arbitrarily broken up without regard to data boundaries.

In the command and response descriptions in this section, <`sec_flag`> is only indicated when it can be a value other than '0'.

Message Buffer Sizes

Every message has a fixed length header, a variable length body, and a fixed length trailer. The message format and number of bytes in each element is shown in Table 20.

Table 20. Message Element Sizes

	Header (fixed size = 5)				(variable size)	Trailer (fixed size = 3)	
Element	<SOH>	<protocol_id>	<seq_flag>	<req_cmd | resp_cmd>	<body>	<checksum>	<EOT>
Size	1	1	1	2	<108 max>	2	1

The maximum message buffer size is 116 (decimal) bytes, including header and trailer. The total number of header + trailer bytes = 8. Max <body> size = 108 bytes. The number of bytes in the message body varies from command (or response) to command (or response). For a given command or response, some are always the same number of bytes and others contain a variable amount of information.

Checksum Calculation Algorithm

The `BK,I` command is used in the example below:

Table 21. Checksum Calculation Table

Byte Description	Byte	Byte in Hex (as sent in message)
SOH	SOH	01
ProtId	N	4e
SeqNbr	0	30
ReqCmd	B, K	42, 4b
HT	HT	09
Param	I	49
Cksum 0	**TBD**	**TBD**
Cksum1	**TBD**	**TBD**
EOT	EOT	04

The checksum is calculated on those items following `SOH` and before the inserted checksum value. The calculation is the negative sum mod 256 of those values.

4e+30+42+4b+09+49=15d

Mod 256 of 15d = 5d

To negate that value: (+ff)+(-5d)+(+01)=a3

The `a3` checksum value is converted to two hex digits and inserted in the message as shown in Table 22.

Table 22. Checksum Value Conversion to Byte

Byte Description	Byte	Byte in Hex
SOH	SOH	01
ProtID	N	4e
SeqNbr	0	30
ReqCmd	B, K	42, fb
HT	HT	09
Param	I	49
Chksum0	**A**	**41**
Chksum1	**3**	**33**
EOT	EOT	04

Naming Conventions (_name)

Names of devices appear in the command descriptions (examples include `src_name, dest_name`). The following conventions apply to device names:

- Names must be from 1 to 8 printable ASCII characters.

  ASCII characters ? and * are excluded, and must not be used. The Series 7000 does not truncate a name longer than eight characters, but declares it an invalid name in its error response.

- Names are case sensitive.

  If all upper case letters are used in a in a name, NODE_CON for example, the system will acknowledge the name, Node_Con, as invalid or as a different object because the case of the letters do not match.

- Spaces in names are discouraged.

  Use an underscore instead of a space, to avoid confusion and keep all characters visible. For example, use VTR_17 rather than VTR 17.

Parameter Quantity (nbr_)

Various parameters describing quantities appear in the command descriptions (examples include `nbr_entries`, `nbr_sources`). These are the ASCII representation of hex quantities, transmitted with the most significant digit first. Normally, only the number of characters necessary to express the size of the number are sent. These are not fixed size fields.

Level Bitmap

In the descriptions, `level_bitmap` appears frequently. A level_bitmap is a 32 bit quantity where each bit represents the presence (=1) or absence (=0) of a particular level for that command or response. The least significant bit (right-most) represents Level #0; the most significant bit (left-most) represents Level #31. The `QN,L` command allows the user to find out the Level Names for each of the bits. We are using `level_bitmap` instead of `level_names` in our commands and responses because it significantly reduces the message buffer sizes needed for the protocol. A 32-bit `level_bitmap` is translated into an equivalent representation of 8 hex digits (= 0,...9,A,...F). These 8 hex digits are then converted to ASCII (='0',...'9', 'A',...'F') and sent with the most significant byte first. The hex digits 'A'...'F' can be sent as upper or lower case ('a'...'f').

For example, the 8-character ASCII message `0000047d` translates into the following bitmap rendering:

```
0000 0000 0000 0000 0000 0100 0111 1101
```

Checking for level presence or absence from 0 through 31 (right to left), this bit map indicates that levels 0, 2, 3, 4, 5, 6, and 10 are present in this command or response.

Refreshing Protects

If a native protocol control port protects particular destinations on the Series 7000, the port is responsible for refreshing those protects periodically or the protects will be dropped by the Series 7000.

The refresh interval can be disabled (=0), or set ≤ 255 seconds. If a Level 4 command (any Level 4 command) is not received with at least this periodicity, the native protocol port decides that the external device is no longer active and sends a device-delete message to the system. As a result, all protects currently held by this native protocol port are dropped.

Refer to the `BK,I` command description which allows the device to query and set the refresh interval. This can also be set from the Graphic User Interface. The `BK`, with no parameters, has no side effects and can be used to keep protects refreshed in the absence of other Level 4 command activity.

Level 2 `ACK`s from the external native protocol device do not reset the timeout counter.

Error Codes

Three sources of error codes may be returned to the external device as a result of a command transmitted to the Series 7000 System:

- Level 2 NAK errors
- Level 4 Errors
- Level 4 MCPU Errors.

Level 2 NAK Errors

Negative Acknowledgement (NAK) related error codes are generated by Native Protocol Level 2 using the RS-232,RS-422 interface. These errors are not present for the Ethernet Native Protocol interface.

An example of a Level 2 error is a Time Out Error. Time out interval begins upon reception of an SOH and is halted at the reception of an EOT. Time out interval is one (1) second for data rates from 2400 to 38.4k Baud. For slower rates, time out is equal to 2400/data rate = time out in seconds. For example, at 300 Baud, time out = 8 seconds (2400/300 = 8).

Level 2 (NAK) Error Code Descriptions

The following codes are sent with NAK's from the 7000 to the external device. The external device is also responsible for NAKing the 7000 when appropriate, but does not have to rigidly adhere to sending these error codes. However, if specific errors are reported with an external device's NAK, they should be defined as:

<NAK> <error_code> <EOT>

<error code> is defined as a Hexadecimal number 71 - 79 (Table 23).

Table 23. Native Protocol Level 2 NAK Error Codes

Decimal Value	Hex Value	Meaning	Description
113	71	Buffer Size Exceeded	The number of characters received since the last detected SOH is greater than the maximum Native Protocol message length.
114	72	Buffer Not Available	No buffer within the input queue was empty and able to store the incoming message. Buffer not available is only reported after an otherwise good message is received.
115	73	Undefined	Undefined
116	74	Chip Level Error	Parity, overrun, etc. Error detected by UART. This error is currently not sent by the 7000. However, the external device may use this code to report a NAK to the 7000.

Table 23. Native Protocol Level 2 NAK Error Codes - (continued)

Decimal Value	Hex Value	Meaning	Description
117	75	Checksum Error	Packet had a bad checksum. Low Level errors such as framing, overrun, etc., are reported as checksum errors.
118	76	Time Out Error	Time out interval is begun upon the reception of an SOH, and is halted at the reception of an EOT. Time out interval is one 1) second for data rates from 2400 to 38.4k Baud. For slower rates, time out is equal to 2400/data rate = time out in seconds. For example, at 300 Baud, time out = 8 seconds (2400/300 = 8).
119	77	Missing SOH	No SOH detected in message. Results when EOT is detected without a preceding SOH.
120	78	Missing EOT	No EOT detected in message. Results when two SOHs are detected without an EOT between. For this to occur, the receiver would have to miss the EOT of the first message and the sender would have to send the second message either without receiving a proper acknowledgment for the first packet, or receiving an erroneous ACKnowledge for the packet whose EOT went undetected. This is an unlikely error. Most probably, when an EOT is missed by a receiver, a time out will occur within the receiver while it is waiting for the rest of the packet (and therefore, the missed EOT).
121	79	Amezi On Hold	The Amezi that is processing Native Protocol requests has been placed on hold, and is not currently accepting any messages. This is usually only encountered in a system configured for redundant Amezis with dedicated data lines to each, and is a signal to the external controller to attempt to resend a message to the other Amezi.

Native Protocol Level 4 Errors

Native Protocol Level 4 Errors can occur when the command is parsed. Examples of error are: unknown command, unknown destination name. Errors of this type are returned to the external device in `ER,nn` responses, described elsewhere in this document.

Error codes of this type are documented in the responses to the `QE` command. They can also be interpreted from the System Diagnostic terminal with the command: `npPrintErrRec 0xhh`, where `hh` is the hex error code. See *Appendix B–Level 4 Error Codes*.

Retrieve an explanation of the numeric error code in one of the following three ways:

Level 4 Error Explanation Retrieval Method 1

Program the controller to request a list of all Level 4 error codes and definitions using the command string:

`<SOH> N Ø Q E E C <EOT>`

The 7000 will respond with separate messages, each containing an error code and a description. Each message must be acknowledged before the next message will be transmitted. To acknowledge a message enter `<ACK>`.

Copy or print the error listing so that the explanations are at hand when error codes are received.

Level 4 Error Explanation Retrieval Method 2

Use the QE command with specific error code parameters.

Level 4 Error Explanation Retrieval Method 3

From the Series 7000 System Diagnostic Terminal, type:

```
npPrintErrRec ØXhh
```

(replace hh with the hex error code you are inquiring about)

For more information on using the Diagnostic Terminal to examine error messages, refer to the Grass Valley Series 7000 product manuals.

If the system is reporting Level 4 error messages resulting from take commands, check the **Cfgd Native Protocol Inet** menu item in the GUI. If the system is trying to control a level that is not enabled in the list of Controllable Levels, an error message will be generated.

MCPU Level 4 Directed Response Error Messages

Certain Native Protocol commands (for example, TA Request Take) are reformatted by Level 4 and passed on to the MCPU for execution. Errors may be generated at this level by the MCPU. Error messages are generated within the MCPU, and returned to the device. When an error occurs, it is sent from the MCPU to Native Protocol level, where it is reformatted into an ER,01 response to the external device. These messages are identified by unique numbers different from those used by Amezi Level 4 error messages.

The error codes returned with these errors are identified by the MC command, as described in *Level 4 Error Message Format* on page 72. Error messages generated by the MCPU are sent to the Amezi via a Directed Response Message. Such messages will be passed on to the external device, as described below. Hex value 01 is the Level 4 error code allocated to Directed Response errors. Associated with each Directed Response message is a secondary code, which defines the error detected by the MCPU, and which defines the content of the rest of the message. This code and its associated data are passed on to the external device.

The format of this message to the external device is as follows (actual text may differ from that shown):

```
<response_cmd=”ER”>
<error_code=0x01>
<request_cmd=”MC”>
<data_1=secondary_code_text_descr>
```

The secondary code is prefixed with a two-digit decimal number that facilitates rapid parsing by the external device (Table 24).

Table 24. Secondary Code

Secondary_Code	Additional Parameters Returned
10 bus_protect	dst_name, level_bitmap
20 src_exclusion	src_name, dst_name
21 prot_denied	dst_name, level_bitmap
22 unprot_denied	dst_name, level_bitmap
23 prot_status	dst_name, level_bitmap
25 salvo_exclusion	TBD
39 no_xpt	src_name, dst_name, level_bitmap
31 no_levels	src_name, dst_name
33 no_assign	none
32 tieline_busy	none
40 other_error	TBD

Native Protocol Messages

Available Two letter Commands

Native Protocol two letter interface commands with brief descriptions are shown in Table 25.

Table 25. Two Letter Interface Commands

Command	Meaning	Description
AS	Machine Assign	Assigns a machine (router input) to control by a specified destination.
BK	Background Activities	Used without parameters to synchronize communications. Used to periodically refresh Protects. Used as a diagnostic tool.
CH	Request Chop	Initiates chopping between specified sources.
CT	Request Clear Destination Tielines	Removes Tieline in Use status if the specified destination is being fed by a tieline.
DA	Machine De–Assign	Removes a machine (router input) from control by a specified destination.
PR	Request Protect	Protects a specific destination from having its source changed.
QA	Query Machine Assignment Status	Checks machine assignment status changes.
QC	Query Combined Destination Status	Returns source status on combined levels of the destination.
QD & Qd	Query Destination Status	Checks sources assigned to destinations by destination name.
QE	Query Error Definition	Retrieves text describing a particular error code.
QI & Qi	Query Destination Status by Index (Response Type 1)	Checks sources assigned to destinations by specific Destination and Level Index.
QJ & Qj	Query Destination Status by Index (Response Type 2)	Checks sources assigned to destinations by Destination Index for all levels.
QL & Ql	Query Destination Status with TieLine Info	Checks sources assigned to destinations by destination name, includes TieLine Information.
QN	Query Names	Checks names associated with Sources, Destinations, Levels, Salvos, Rooms, or TieLines.
QT	Query Date and Time	Checks system date and time information.
QV	Query Salvo Status	Checks sources, destinations, and levels associated with a specified Salvo (Timed Salvo info is not available).
ST	Request Set Date and Time	Sets system date and time information.
TA	Request Take	Takes Sources (on specified levels) to specified destination, by name rather than index.
TD	Request Take Destination	Takes same source to all or specified levels.
TI	Request Take Index with Level Index	Takes Source (on specified level) to specified destination, by index rather than name.
TJ	Request Take Index with Level Bit Map	Takes Sources (on specified levels) to specified destination by index rather than name. Allows Breakaways.
TM	Request Take Monitor Destination	Takes Sources (on specified levels) to the Monitor Destination.
TS	Request Take Salvo	Executes the specified Salvo.
UP	Request UnProtect	Removes Protect from specified Destination.

AS - Machine Assign

Command

```
AS,dest_name,src_name
```

Response

(None)

BK - Background Activities

The BK command can be used to synchronize communications between the external controller and the control port. The external controller sends BK messages (with no parameters) until it receives an ACK from the control port. At this point, communications are synchronized. Any native protocol command can be sent to accomplish synchronization. However, the BK command with no parameters has no side effects.

The BK command can also be used to periodically refresh protects.

The BK command also has diagnostic uses. When the BK command is sent with optional parameters, information described below is returned to the external device.

Command

```
BK [,parameter [,mask]]
```

A maximum of one `parameter` can be specified per BK call. Each parameter consists of a single, case sensitive letter, defined below.

Table 26. BK Command Parameters

Parameter	Description
N	Return System Name
R	Return Protocol Processor Software Revision #
T	Return smsApp software title (for stricter version verification)
t	Return native protocol software title (for stricter version verification)
F	Return set of flags defining reset occurrences, protects dropped, name/ID table updates, etc. (bit flag = 1 if defined change occurred — see below for change definitions)
f	Mask clear change flags defined in 'mask' (see below for mask definition). mask bits = 1 indicate flags to be cleared.
D	Clears the flags associated with the `QD,no_parameter` command. After `BK,D` is sent, the next `QD,no_parameter` command will result in destination statuses for all destinations being returned.
A	Clears the flags associated with the `QA,no_parameter` command. After `BK,A` is sent, the next `QA,no_parameter` command will result in all assignment statuses being returned.

Table 26. BK Command Parameters - (continued)

Parameter	Description
P	Returns port configuration parameters fixed by Graphic User Interface configuration, and which cannot be modified by the native protocol device.
I	[, < RefreshIntervalInSecs | OFF=0 >} Sets or returns Refresh Interval. If no interval is specified, this is interpreted as a request to simply return the existing value. Refresh Interval is specified to be <= FF hex.
E	[, < ON | OFF >] Sets or returns status of Level 4 Echo (err = 00). If no parameter is specified, this is interpreted as a request to simply return the existing status.
d	Returns the name of this port or device.
2	Null command. This message is processed entirely by Level 2, and is not passed up to Level 4. If the message has the correct checksum, an ACK will be returned to the external controller. The intended use for this command is to allow an external controller to verify that a redundant, backup control port is alive, without side effects occurring in the Level 4 code. It can also be used on the active control port, but this is not its intended use.

Response

(No response if no returned data.)

Response with returned data is:

`KB,parameter,data`

`parameter` consists of a single, case sensitive letter, defined in Table 27:

Table 27. KB Response Parameters

Parameter	Data
N	system_name_string
R	Protocol_Processor_software_rev#_string
T	smsApp_title
t	native_title
I	Refresh_Interval_in_Seconds. If = 0, refresh is not enabled. A value will be returned for both set and query requests (see **`I`** in Table 26).
E	Echo = ON | OFF. A value will be returned for both set and query requests (see **`E`** in Table 26).
d	Device_Name
P	PnlLck=<ON|OFF>, ChopLck=<ON|OFF>, SlvLck=<ON|OFF>, ProtOvrd=<ON|OFF>, MonCtl=<ON|OFF>, CtlblLvls=lvlBitMap
D	none
A	none
2	none
F	Four ASCII characters representing four HEX digits (16 bits), with bits flagging changes since flags last cleared by **`f`** parameter. Most significant hex digit is sent first (that is, b15...b12). See Table 28.

KB, F Response

For a `KB, F` response, the `data` is defined in Table 28:

Table 28. KB Command Response Bits

Bit #	Meaning
0 (lsb)	Reset has occurred. When reset occurs, this bit is set =1; all other flag bits are also set = 1.
1	any protects initiated by this control port were dropped.
2	destination changes
3	tieline changes
4	source changes
5	level changes
6	salvo changes
7	room changes
8 - 14	(Reserved. Always returned = 1 unless user has cleared these bits.)
15	Redundant switchover event occurred.
f, mask	(none)

Here is an example of the use of BIT flags for parameters = `f` or `F`.

The user queries Native Protocol using `BK,F` — and receives a reply with parameter bit #4 set to indicate that there has been a change (addition, modification, deletion) to the system source name table. The user next uses the `BK,f`, mask (where mask bit #4 = 1) command to clear bit #4 from the mask. The next `QN,S` command results in all source names being downloaded to the user. The user again queries using `BK,F`. The reply shows that bit #4 = 0, indicating that the source name list just downloaded is current.

Do not confuse flags discussed with `f` & `F` options with those discussed with `D` & `A` options. The `QD` and `QA` commands allow the user to download incremental changes in Destination and Assignment Status tables. The `D` and `A` options of the `BK` command allow the bit arrays that keep track of these incremental changes to be cleared. This allows re-synchronizing with the 7000 data base (for Destination & Assignment Status) if, for some reason, the external device resets.

Using a server timeout value (configured Refresh Rate or `BK,I` command) of zero is not recommended. If a connection is broken or a client crashes, the MCPU may not close the socket for a long period of time.

CH - Request Chop

Command

The format of the Chop command is identical to that of the `TA` Request Take command.

```
CH,dest_name,nbr_sources,src_name_entry1,
[..., src_name_entryn]
```

However, the specified source names to be taken to the destination actually specify the chop source names and levels. This command results in chopping with the already established destination status.

To specify a chop operation, first Take sources and levels to a destination, then Chop to the same destination with the chop sources and levels.

Response

(none)

Command

(<seq_flag='0'> for the last sequence sent.)

`dest_name`	Destination to be taken to
`nbr_sources`	Number of following entries (must be at least one)

`src_name_entryn` is defined as:

```
src_name,level_bitmap
```

Response

(none)

CT - Clear Tielines

Command

```
CT,dest_name
```

Response

(none)

DA - Machine De-assign

Command

```
DA,dest_name,src_name
```

Response

(none)

PR - Request Protect

Command

```
PR,dest_name,level_bitmap
```

Response

No direct response, but a successful PR command will return a `Directed Response Message = prot_status` from the MCPU, which in turn results in a message to the external device `ER`, `error_code = 01`, echoed command = `MC`.

QA - Query Machine Assignment Status

Command

```
QA[,dest_name]
```

`dest_name` This can be either:

-An ASCII name of up to 8 characters. In this case the assignment status for a single destination will be returned, or

-Blank. If so, assignment status is returned for all destinations for which assignment has changed since the last time information was sent. Destination assignment information is returned for all levels (for the changed as well as unchanged levels) to which a machine is assigned. The BK,A command can be used to force return of all destination assignments.

Response

```
AQ,dest_name,nbr_sources[,src_name1,...,src_namen]
```

(<seq_flag='0'> for the last sequence sent.)

`nbr_sources`	Number of sources assigned to that destination which are being reported in this message sequence. If there are no sources assigned to this destination, `nbr_sources = 0` will be returned.

`QA` (with no dest_name specified) is one of several commands whose response can consist of more than one message sequence. `QA` with no parameter can result in a sequence of messages for each of many destinations. To help the external device determine that a particular message sequence is the very last such sequence, it will be followed by a Level 4 Echo (ER,ØØ), if that Port Configuration Feature is enabled. `QA` (with a dest_name specified) is not followed by a Level 4 Echo.

QC - Query Combined Destination Status

Command

`QC[,dest_name]`

`dest_name`	This can be either: - An ASCII name of up to 8 characters. In this case the status information for a single destination will be returned, or - Blank. If so, destination status information is returned for all destinations for which status has changed since the last time status information was sent. Destination status information is returned for all levels (for the changed as well as unchanged levels) which have status. The `BK,D` command can be used to force return of all destination status.

If there are no sources currently on some of the levels defined for the destination, no information is reported for those levels.

Response

`CQ,dest_name,dst_level_bitmap[,src_name_entry1,...,src_name_entryn]`

(<seq_flag='0'> for the last sequence sent.)

`dst_level_bitmap`	Describes the levels configured for destination. For source connected to first level dst_level_bitmap=0 will be returned if no source is connected.

src_name_entry1 is defined as:

```
<'N'|'P'>,<'N'|'C'>,srcm_name,level_bitmap,
[device_name], [chop_src_name]
```

Parameters for this response are explained in Table 29.

Table 29. QC Response Parameters

Parameter	Meaning
'N' |'P'	Not-protected or Protected
'N' | 'C'	Not-chopping or Chopping
src_name	One of the sources currently taken to the destination
level_bitmap	Describes the levels of that destination which the source is on
device_name	The device currently holding the protect. If the destination is not protected, or the device name is unknown, the field is left blank.
chop_src_name	The name of the source chopping to this destination

QC (with no dest_name specified) is one of several commands whose response can be more than one message sequence. QC with no parameter can result in a sequence of messages for each of many destinations. To help the external device determine that a particular message sequence is the last sequence, it is followed by a Level 4 Echo (ER,ØØ), if that Port Configuration Feature is enabled. QC (with a specified dest_name) is not followed by a Level 4 echo.

QD - Query Destination Status

Command

```
QD[,dest_name]
```

dest_name	This can be either: - An ASCII name of up to 8 characters. In this case the status information for a single destination will be returned, or - Blank. If so, destination status information is returned for all destinations for which status has changed since the last time status information was sent. Destination status information is returned for all levels (for the changed as well as unchanged levels) which have status. The BK,D command can be used to force return of all destination status.

If there are no sources currently on some of the levels defined for the destination, no information is reported for those levels.

Response

```
DQ,dest_name,nbr_sources[,src_name_entry1,...,
src_name_entryn]
```

(<seq_flag='0'> for the last sequence sent.)

`nbr_sources`	Number of sources on that destination which are being reported in this message sequence. If there are no sources on this destination at any of its levels, nbr_sources = 0 will be returned.

`src_name_entryn` is defined as:

```
<'N'|'P'>,<'N'|'C'>,src_name,level_bitmap,
[device_name], [chop_src_name]
```

Data for this response is identical to the `QC` command response described previously.

`QD` (with no dest_name specified) is one of several commands whose response can be more than one message sequence. `QD` with no parameter can result in a sequence of messages for each of many destinations. To help the external device determine that a particular message sequence is the last sequence, it is followed by a Level 4 Echo (ER,ØØ), if that Port Configuration Feature is enabled. `QD` (with a specified dest_name) is not followed by a Level 4 echo.

Qd - Query Destination Status

Command

The `Qd` command is the same format as the `QD` command:

```
Qd[,dest_name]
```

Response

The response to the `Qd` command is similar to that of the `QD` command, with the following differences:

- The response Command is `dQ`.
- The src_name returned will be `NO_XPT` if that condition applies to the particular set of crosspoints being reported.

QE - Query Error Definition

Error messages returned through the controller channels are terse and identified by a two byte code. This command allows the user to retrieve the text describing Level 4 error codes. Level 2 error codes (associated with NAKs) are documented elsewhere.

This facility was provided so that error code interpretation could be determined by the on-line controlling device, without having to look up codes in possibly outdated documentation. Error code definitions can always be determined in the latest software release.

Command

```
QE,error_code
```

`error_code`	This can be either: - Any 2 hex digit (represented by ASCII) error code received in an ER,nn message in response to a command sent to the Amezi, or - Blank. In this case, error definition information will be returned for all error codes.

Response

```
EQ,error_code,error_definition_string
```

(<seq_flag='0'> for the last sequence sent.)

QI - Query Destination Status By Index

Command

```
QI,destIndex,lvlIndex
```

`destIndex` and `lvlIndex` are always required

Response

```
IQ,destIndex,lvlIndex,<'N'|'P'>,<'N'|'C'>,srcIndex,
[chop-SrcIndex]
```

This command allows access to destination information by using the destination index vs. the destination name. Similarly, information is returned by index reference vs. name. The returned indexes will always be four ASCII hex characters with leading zeros as needed.

All indexes (`dstIndex`, `lvlIndex`, `srcIndex`) are zero–based hex numbers. These indexes refer to the logical index for each entity, not to the physical port number. However, the system can be configured so that port numbers are identical to index numbers. There is no disconnect index

number, all indexes refer to configured entities. Valid indexes and their association with specific names can be determined using the commands: `QN, ID; QN, IS; QN, L`.

Qi - Query Destination Status By Index

Command

The `Qi` command is the same format as the `QI` command.

```
Qi,destIndex,lvlIndex
```

Response

The response to the `Qi` command is similar to that of the `QI` command, with the following differences:

- The Response Command is `iQ`.
- The `srcIndex` returned will be `0xfffe` if an error condition applies to the crosspoint being reported.

QJ - Query Destination Status By Index

Command

```
QJ[,dest_index]
```

The `dest_index` field may be left blank. If so, destination status is returned for all destinations for which status has changed since the last status information was sent. Destination status information is returned for all levels (changed or unchanged) which have status. The `BK,D` command can be used to force return of all destination status.

If there are no sources currently on some of the levels defined for the destination, no information is reported for those levels.

Response

```
JQ,dest_index,nbr_sources[,src_name_entry1,...,
src_name_entryn]
```

(<seq_flag='0'> for the last sequence sent for a particular destination status.)

`nbr_sources`	Number of sources on that destination which are being reported in this message sequence. If there are no sources on this destination at any of its levels, nbr_sources = 0 will be returned.

`src_name_entryn` is defined as:

```
<'N'|'P'>,<'N'|'C'>,src_index,level_bitmap,
[device_name], [chop_src_index]
```

Parameters for this response are explained in Table 30.

Table 30. QJ Command Response Parameters

Parameter	Meaning
'N' |'P'	Not-protected or Protected
'N' | 'C'	Not-chopping or Chopping
src_index	One of the sources currently taken to the destination
level_bitmap	Describes the levels of that destination which the source is on
device_name	The device currently holding the protect. If the destination is not protected, or the device name is unknown, the field is left blank.
chop_src_name	The name of the source chopping to this destination

Returned indexes will always be four ASCII hex characters with leading zeroes as needed. All indexes (`dstIndex`, `lvlIndex`, `srcIndex`) are zero-based hex numbers. These indexes refer to the logical index for each entity, not to the physical port number. However, the system can be configured so that port and index numbers are identical. There is no disconnect index number; all indexes refer to configured entities. Valid indexes and specific name associations can be determined using the commands: `QN, ID; QN,IS; QN,L.`

The response to `QJ` (with no dest_index specified) can be more than one message sequence. `QJ` with no parameter can result in a sequence of messages for each of many destinations. To help the external device determine that a particular message sequence is the last, it is followed by a Level 4 Echo (ER,ØØ), if that Port Configuration Feature is enabled. `QJ` (with a specified dest_index) is not followed by a Level 4 echo.

Qj - Query Destination Status By Index

Command

The `Qj` command is the same format as the `QJ` command.

```
Qj[,dest_index]
```

Response

The response to the `Qj` command is similar to that of the `QJ` command, with the following differences:

- The Response Command is `jQ`.
- The `srcIndex` returned will be `0xfffe` if that condition applies to the particular set of crosspoints being reported.

QL - Query Destination Status With Tieline Info

Command

Allows access to destination information, with possible tie-line information attached. The format of the request is the same as for the `QD` command.

`QL[,dest_name]`

`dest_name`	This can be either: - An ASCII name of up to 8 characters. In this case the status information for a single destination will be returned (seq_flag = 0), or - Blank. If so, destination status information is returned for all destinations for which status has changed since the last time status was sent. Destination status information is returned for all levels (for changed as well as unchanged levels) which have status. The `BK,D` command can be used to force the return of all destination status.

If there are no sources currently on some of the levels defined for the destination, no information is reported for those levels.

Response

```
LQ,dest_name,nbr_sources[,src_name_entry1,...,
src_name_entryn]
```

(<seq_flag='0'> for the last particular destination status report.)

`nbr_sources`	Number of sources on that destination which are being reported in this message sequence. If there are no sources on this destination at any of its levels, nbr_sources = 0 will be returned.

`src_name_entryn` is defined as:

```
<'N'|'P'>,<'N'|'C'>,<'N'|'S'>,<'N'|'C'>, src_name,
level_bitmap,[device_name],[chop_src_name],
[downstreamSrcName],[downstreamChopSrcName]
```

Data for this response is explained in Table 31.

Table 31. LQ src_name_entry1 Description

Parameter	Meaning
'N' | 'P'	Not-protected or Protected
'N' | 'C'	Not-chopping or Chopping
'N' | 'S'	No downstream source name or downstream source name exists
'N' | 'C'	No downstream chop source name or downstream chop source name exists
src_name	One of the sources currently taken to the destination
level_bitmap	Describes the levels of that destination which the source is on
device_name	The device currently holding the protect. If the destination is not protected, or the device name is unknown, the field is left blank.
chop_src_name	Name of the source chopping to this destination
downstreamSrcName	Name of the downstream source, if any
downstreamChopSrcName	Name of the downstream chopping source, if any

`QL` (with no `dest_name` specified) is one of several commands whose response can be more than one message sequence. `QL` with no parameter can result in a sequence of messages for each of many destinations. To help the external device determine that a particular message sequence is the very last such sequence, it will be followed by a Level 4 Echo (ER,ØØ), if that Port Configuration Feature is enabled. `QL` (with a specified dest_name) is not followed by a Level 4 echo.

Ql - Query Destination Status With Tieline Info

Command

The `Ql` command is the same format as the `QL` command.

```
Ql[,dest_name]
```

Response

The response to the `Ql` command is similar to that of the `QL` command, with the following differences:

- The Response Command is `lQ`.
- The `src_name` returned will be `NO_XPT` if that condition applies to the particular set of crosspoints being reported.

QN - Query Names

Command

`QN,parameter`

One parameter can be specified per `QN` call. The parameters available, and their meaning, are listed in Table 32.

Table 32. QN Parameters

Parameter	Meaning
S	Source
D	Destination
L	Level
V	Salvo
R	Room
T	Tie line
M	
Y	
IS	Sources with source indexes
ID	Destinations with destination indexes

S Response

`NQ,S,nbr_sources[,src_name_entry1,...,src_name_entryn]`

(<seq_flag='0'> for the last sequence sent.)

`src_name_entryn` is defined as:

`src_name,<'N'|'T'>,level_bitmap`

`<'N'|'T'>` Indicates No-TieLine or Tieline related

D Response

`NQ,D,nbr_destns[,dest_name_entry1,...,`
`dest_name_entryn]`

(<seq_flag='0'> for the last sequence sent.)

`dest_name_entryn` is defined as:

`dest_name,<'N'|'T'>,level_bitmap`

`<'N'|'T'>` Indicates No-TieLine or Tieline related

L Response

`NQ,L,nbr_levels[,lvl_name_entry1,...,lvl_name_entryn]`

(<seq_flag='0'> for the last sequence sent.)

lvl_name_entryn is defined as:

```
lvl_name,lvl_number,< 'R' | 'N' >
```

lvl_number	A hex ASCII number from 00 to 1f (representing levels 00 to 31 decimal)
'R' | 'N'	Specifies that this level is Restricted or Not with respect to assignments

V Response

```
NQ,V,nbr_salvos[,salvo_name1,...,salvo_namen]
```

(<seq_flag='0'> for the last sequence sent.)

R Response

```
NQ,R,roomName,<'1'|'2'|'3'>,nbrDestNames
[,destName1,...,  destNamen]
```

<'1'|'2'|'3>	Indicates the Room Class Type

There is one message returned per room Name.

nbrDestNames	Value is <= 8.

The information for each room Name fits within one message buffer.

(<seq_flag='0'> for the last sequence sent.)

T Response

```
NQ,T,nbr_tieLines[,tieLine_entry1,...,tieLine_entryn]
```

(<seq_flag='0'> for the last sequence sent.)

tieLine_entryn is defined as:

```
tieLineName,<'N'|'R'>,beginDstName,
upstreamLvlName,endSrName, downstreamLvlName
```

<'N'|'R'>	indicates Not-reserved or Reserved

M Response

```
NQ,M,nbr_tieline_entries[,tieline_entry1,...,
tieline_entryn]
```

(<seq_flag='0'> for the last sequence sent.)

tieline_entry is defined as:

```
tieline_Name,<'N'|'R'>,tlTypeName,beginDstName,
endSrcName
```

Y Response

```
NQ,Y,nbr_tltype_entries[,tltype_entry1,...,
tltype_entryn]
```

(<seq_flag='0'> for the last sequence sent.)

`tltype_entryn` is defined as:

```
tltype_name,lvlbitmap
```

`lvlbitmap` includes the EndLevels in this entry, and for each consecutive 1 in the EndLevels bitmap (least significant bit first), a `lvlbitmap` of the BeginLevels that feed this endLevel. If there are many EndLevels in the tieline type, it may take more than one `tltype_entry` to send them all.

IS Response

```
NQ,S,nbr_sources[,src_name_entry1,...,src_name_entryn]
```

(<seq_flag='0'> for the last sequence sent.)

`src_name_entryn` is defined as:

```
src_name,src_index,<'N'|'T'>,level_bitmap
```

`<'N'|'T'>`	No-tieline or Tieline Related
`src_index`	Four digit ASCII hex # with leading Ø's

ID Response

```
NQ,D,nbr_destns[,dest_name_entry1,...,
dest_name_entryn]
```

(<seq_flag='0'> for the last sequence sent.)

`dest_name_entryn` is defined as:

```
dest_name,dest_index,<'N'|'T'>,level_bitmap
```

`<'N'|'T'>`	No-tieline or Tieline Related
`dest_index`	Four digit ASCII hex # with leading Ø's

QT - Query Date & Time

Command

```
QT
```

Response

```
ST,yyyymmddhhmmss
```

hh is 00...23

QV - Query Salvo Status

Command

`QV,salvo_name`

Timed salvos are not included in the `QV` command.

Response

`VQ,salvo_name,nbr_entries[,entry1,...,entryn]`
(<seq_flag='0'> for the last sequence sent.)

`entryn` is defined as follows:

`dest_name,src_name,level_bitmap`

ST - Request Set Date & Time

Command

`ST,yyyymmddhhmmss`

The parameters are in ASCII (Table 33).

Table 33. Date and Time Values

Parameter	Values
yyyy	1993...2999
mm	01...12
dd	01...31
hh	00...23
mm	00...59
ss	00...59

Response

(none)

TA - Request Take

Command

`TA,dest_name,nbr_sources,src_name_entry1,...,`
`src_name_entryn]`
(<seq_flag='0'> for the last sequence sent.)

`dest_name` Destination to be taken to

`nbr_sources` Number of following entries (must be at least one)

`src_name_entryn` is defined as:

```
src_name,level_bitmap
```

Response

(none)

TD - Request Take Destination

Command

`TD,dest_name,src_name_entry`

(<seq_flag='0'> for the last sequence sent.)

`dest_name` Destination to be taken to

`src_name_entryn` is defined as:

```
src_name[,levelbitmap]
```

With no level bitmap specified, the source will be taken to all levels of the destination.

Response

(none)

TI - Request Take Index With Level Index

Command

`TI,destIndex,srcIndex[,levelIndex]`

This command allows a Take Request to be specified using indexes vs. names. If no level Index is specified, then an all-level take occurs.

All indexes (`dstIndex`, `lvlIndex`, `srcIndex`) are zero–based hex numbers. These indexes refer to the logical index for each entity, not to the physical port number. However, the system can be configured so that port numbers are identical to index numbers. There is no disconnect index number, all indexes refer to configured entities. Valid indexes and their association with specific names can be determined by using the commands: `QN, L; QN, ID; QN, IS`.

Response

(none)

TJ - Request Take Index With Level Bitmap

Command

```
TJ,dest_index,nbr_sources,src_name_entry1,...,
src_name_entryn]
```

(<seq_flag='0'> for the last sequence sent.)

`dest_index`	Destination to be taken to
`nbr_sources`	Number of following entries (must be at least one)

`src_name_entryn` is defined as:

```
src_index,level_bitmap
```

All indexes (`dstIndex`, `lvlIndex`, `srcIndex`) are zero–based hex numbers. These indexes refer to the logical index for each entity, not to the physical port number. However, the system can be configured so that port numbers are identical to index numbers. There is no disconnect index number. All indexes refer to configured entities. Valid indexes and their association with specific names can be determined by using the commands: `QN,L; QN,ID; QN,IS`.

Response

(none)

TM - Request Take Monitor Destination

Command

```
TM,dest_name
```

Response

(none)

TS - Request Take Salvo

Command

```
TS,salvo_name
```

`salvo_name`	An untimed salvo

Response

(none)

UP - Request Unprotect

Command

`UP,dest_name,level_bitmap`

Response

No direct response. However, a successful `UP` command will result in the return of a `Directed Response Message = prot_status` from the MCPU, which in turn results in a message to the external device `ER`, error_code = 01, echoed command = `MC`.

Section 4

Serial Node Controller Protocol

Introduction

This section describes the format of the Series 7000 Serial Node Controller (SNC) RS-485 Pro-Bel SW-P-02x (extended) protocol. Some additional information about the Pro-Bel SW-P-02 standard protocol is also included.

This protocol is designed to allow an Omnibus system to control a Series 7000 Node Controller directly, using the established Pro-Bel protocol and this extension to that protocol. The extension allows larger Series 7000 messages (in SSML protocol format) to be transferred to and from the Series 7000 Node Controller. Basic matrix control can be accomplished using the standard Pro-Bel 02 protocol, and so use of the Pro-Bel 02x protocol is not required but can add functionality.

SSML is a Grass Valley proprietary protocol used to communicate within the Series 7000 system via the Node Control Bus. Formatting and use of Series 7000 SSML messages communicated via Pro-Bel 02x are not discussed in this document. Contact Grass Valley Customer Service if you require this information.

Physical Layer

RS-485

The Serial Node Controller physical interface is RS-485. A serial cable is required to connect the Series 7000 Node Controller to the controlling device. Maximum cable length is determined by customer needs, limited only by the RS-485 standard and the customer site environmental noise.

Link layer

Format Types

Two link layer protocols are defined:

- Pro-Bel SW-P-02 as defined in Pro-Bel's *General Switcher Communications Protocol*, Issue 7 of May 5, 1992. This protocol type is referred to as Type 02, and
- A Pro-Bel SW-P-02x, an extension for communicating Series 7000 Node Controller message types requiring 8 bits. This protocol is referred to as Type 02x.

Link Characteristics

Communication is via a full-duplex RS-485 asynchronous link at:

- 8 data bits,
- 1 stop bit,
- No parity, and
- 38400 baud.

Protocol message types are sent and received in any order as needed. The two protocols are mixed on the link. It is up to the sender to choose the correct protocol type for a particular message, and up to the receiver to be able to correctly parse either message type.

Some messages transmitted to the Series 7000 Node Controller by the controlling device require responses. It is not necessary for the controlling device to wait for a response before transmitting further messages.

Packet Pacing

Pacing is required by the controlling device between packets transmitted to the Series 7000 Node Controller. The method of pacing is to be a delay between the last character of one packet and the first character of the succeeding packet. This value is to be selected at test, but the current best estimate is two character times (at 38400 baud = 500 µs).

Handshaking

There is no link layer handshaking. No acknowledgment or error response is generated by the receiver upon reception of an 02 or 02x packet at the link layer (e.g., `ACK`, `NAK`). If the receiver detects any error (checksum, unexpected byte count, unrecognized command) then the message is ignored. If a `SOM` or `SOMx` is received in the middle of an incoming packet, all the proceeding bytes of the incoming packet are discarded and a new packet is begun.

Success or failure of packet transmission can only be accomplished at the Packet Layer, either by receiving the corresponding response for messages that have responses, or by querying the Series 7000 Node Controller for the information needed for a verification (such as a crosspoint query via an `Interrogate` command). Refer to each message definition to determine what responses, if any, are expected.

The 02 Protocol

02 Message Format

`<SOM> <command> <message> <checksum>`

`<SOM>`	(1 byte) Start of message = 0x0FF
`<command>`	(1 byte) Message type. Also defines message length.
`<message>`	(0-n bytes) Message body.
`<checksum>`	(1 byte) 7 bit, sum all message bytes, excluding `SOM`, Count and any Bit7ESCs. Then compliment the Sum and Increment the sum by value of 1. Apply mask 0x7F (or value with binary 01111111). The result is the message checksum.

Special 02 Protocol Characters

`SOM (0x0FF)`	Signals to the receiver the start of a Type 02 message. Any incoming message is completed when the number of bytes expected are received. Any message not completely received when a `SOM` (or `SOMx`) is received is simply discarded, and the new message signaled by the `SOM`(x) is received.

The 02x Protocol

02x Message Format:

```
<SOMx> <command> <message> <checksum>
```

`<SOMx>`	(1 byte) Start of message = 0x0FE
`<count>`	(1 byte) Message length in bytes. Only includes <message> field. Does not include <SOM>, <count>, or <checksum> fields. Also does not include any 7 bit escape (BIT7ESC == F1) bytes.
`<message>`	(0-120 bytes) Message body. Maximum 120 bytes. Does not include any BIT7ESC bytes.
`<checksum>`	(1 byte) 7 bit, sum all message bytes, excluding SOMx, Count and any Bit7ESCs. Then compliment the Sum and Increment the sum by value of 1. Apply mask 0x7F (or value with binary 01111111). The result is the message checksum.

Special 02x Characters

`SOMx (0x0FE)`	Signals to the receiver the start of a TYPE 02x message. Any incoming message is completed when the number of bytes expected are received. Any message not completely received when a SOMx (or SOM) is received is simply discarded, and the new message signaled by the SOM(x) is received.
`BIT7ESC (0x0F1)`	Signals to the receiver to set the previous byte's most significant bit. For example, a 2 is received in byte 5, then an F1 is received in byte 6. The receiver must then set the m.s.b. of byte 5, changing byte 5's value from 2 to 130. The F1 is then discarded and is not counted towards the COUNT field's message byte count. Note: Series 7000/Node Controller traffic should not require an excess of data greater than 127.

Packet Layer

Link Messages

The following two messages are required to establish communications between a controlling device and a Matrix Controller or a Node Controller. Until these messages are received, the Matrix Controller or Node Controller it will not respond to other messages from the controlling device.

MSG_SET_PRIMARY

Command

Direction:	Controlling Device -> Series 7000 Node Controller
Purpose:	Set the Matrix Controller or Node Controller to primary
Hexadecimal:	FE 02 53 22 0B
Rate:	On demand

Table 34. MSG_SET_PRIMARY Command

Length (5 Bytes)	Element	Description
BYTE	SOMx	= 0x0Fe
BYTE	ByteCount	= 0x002
BYTE	Socket	= 0x053
BYTE	msg Type	= 0x022
BYTE	Checksum	= 0x00B

Response

Direction:	Series 7000 Node Controller -> Controlling Device
Purpose:	Response
Hexadecimal:	FE 05 2B 0E 01 00 01 45

Table 35. MSG_SET_PRIMARY Response

Length (8 Bytes)	Element	Description
BYTE	SOMx	= 0x0Fe
BYTE	ByteCount	= 0x05
BYTE	Socket	= 0x2B
BYTE	msg Type	= 0x0E
BYTE		= 0x01
BYTE		= 0x00
BYTE		= 0x01
BYTE	Checksum	= 0x45

MSG_GET_NC_HEALTH

Command

Direction:	Controlling Device -> Series 7000 Node Controller
Purpose:	Get the health Message from the Matrix Controller or Node Controller
Hexadecimal:	FE 02 54 1A 12
Rate:	On demand

Table 36. MSG_GET_NC_HEALTH Command

Length (5 Bytes)	Element	Description
BYTE	SOMx	= 0x0FE
BYTE	ByteCount	= 0x002
BYTE	Socket	= 0x054
BYTE	msg Type	= 0x01A
BYTE	Checksum	= 0x012

Response

Direction:	Series 7000 Node Controller -> Controlling Device
Purpose:	Response
Hexadecimal:	FE 05 2B 32 00 28 02 79

Table 37. MSG_GET_NC_HEALTH Response

Length (8 Bytes)	Element	Description
BYTE	SOMx	= 0x0Fe
BYTE	ByteCount	= 0x05
BYTE	Socket	= 0x2B
BYTE	msg Type	= 0x32
BYTE		= 0x00
BYTE		= 0x28
BYTE		= 0x02
BYTE	Checksum	= 0x79

The Type 02 Command Set

For reference, the following commands used by the standard Pro-Bel 02 protocol are included in this document. For greater detail, see the Pro-Bel General Switcher Communication Protocol, Issue 7 of May 5 1992 (Appendix 1). All values in the Type 02 commands, excepting the SOM Byte 0x0FF, are specified as decimal values.

Interrogate (1)

Direction: Controlling Device -> Series 7000 Node Controller

Purpose: Request for Destination Status

Rate: On demand

Table 38. Interrogate Command

Length (5 bytes)	Element	Description
Byte	SOM	= 0x0FF
Byte	Command	= 1
Byte	Multiplier: Bit 7 Bit 4-6 Bit 3 BIT 0-2	 = 0 Destination number DIV 128 All main router destinations. = 0 = 0
Byte	Destination	Destination number MOD 128
Byte	Checksum	

Connect (2)

Direction: Controlling Device -> Series 7000 Node Controller

Purpose: Request a crosspoint connect

Rate: On demand

Table 39. Connect Command

Length (6 bytes)	Element	Description
BYTE	SOM	= 0x0FF
BYTE	Command	= 2
BYTE	Multiplier Bit 7 Bit 4-6 Bit 3 BIT 0-2	 = 0 Destination number DIV 128 = 0 = Source number DIV 128
BYTE	Destination	Destination number MOD 128
BYTE	Source	Source number MOD 128
BYTE	Checksum	Calculated

Tally (3)

Direction: Series 7000 Node Controller -> Controlling Device

Purpose: Returns Destination Status in response to an `Interrogate (1)` command

Rate: On demand

Table 40. Tally Command

Length (6 bytes)	Element	Description
BYTE	SOM	= 0x0FF
BYTE	Command	= 3
BYTE	Multiplier Bit 7 Bit 4-6 Bit 3 BIT 0-2	 = 0 Destination number DIV 128 = 0 = Source number DIV 128
BYTE	Destination	Destination number MOD 128
BYTE	Source	Source number MOD 128
BYTE	Checksum	Calculated

Connected (4)

Direction: Series 7000 Node Controller -> Controlling Device

Purpose: Returns Destination Status after a new source has been connected, usually in response to a `Connect (2)` command

Rate: On demand

Table 41. Connected Command

Length (6)	Element	Description
BYTE	SOM	= 0x0FF
BYTE	Command	= 4
BYTE	Multiplier Bit 7 Bit 4-6 Bit 3 BIT 0-2	 = 0 Destination number DIV 128 = 0 = Source number DIV 128
BYTE	Destination	Destination number MOD 128
BYTE	Source	Source number MOD 128
BYTE	Checksum	Calculated

Connect_On_Go (5)

Direction: Controlling Device -> Series 7000 Node Controller

Purpose: Preload Salvo crosspoint

Rate: On demand

Table 42. Connect_On_Go Command

Length (6)	Element	Description
BYTE	SOM	= 0x0FF
BYTE	Command	= 5
BYTE	Multiplier Bit 7 Bit 4-6 Bit 3 BIT 0-2	 = 0 Destination number DIV 128 = 0 = Source number DIV 128
BYTE	Destination	Destination number MOD 128
BYTE	Source	Source number MOD 128
BYTE	Checksum	Calculated

Go (6)

Direction: Controlling Device -> Series 7000 Node Controller

Purpose: Take preloaded Salvo

Rate: On demand

Table 43. Go Command

Length (4)	Element	Description
BYTE	SOM	= 0x0FF
BYTE	Command	= 6
BYTE	Action	= 0 then Set previously received crosspoints = 1 then Clear previously received crosspoints
BYTE	Checksum	Calculated

Connect_On_Go_Acknowledge (12)

Direction: Series 7000 Node Controller -> Controlling Device

Purpose: Response to a `Connect_On_Go (5) command`

Rate: On demand

Table 44. Connect_On_Go_Acknowledge Command

Length (6)	Element	Description
BYTE	SOM	= 0x0FF
BYTE	Command	= 12
BYTE	Multiplier Bit 7 Bit 4-6 Bit 3 BIT 0-2	 = 0 Destination number DIV 128 = 0 = Source number DIV 128
BYTE	Destination	Destination number MOD 128
BYTE	Source	Source number MOD 128
BYTE	Checksum	Calculated

Go_Done (13)

Direction: Series 7000 Node Controller -> Controlling Device

Purpose: Response to a `Go (6) command`

Rate: On demand

Table 45. Go_Done Command

Length (4)	Element	Description
BYTE	SOM	= 0x0FF
BYTE	Command	= 13
BYTE	Action	= 0 then Set previously received crosspoints = 1 then Clear previously received crosspoints
BYTE	Checksum	Calculated

Appendix A

Reference Materials

CheckSum Calculation Code Snippet

This code snippet is provided as one of the ways to calculate the Checksum.

```
/*****************************************************************************

 Function    :    checksumCalc

 Description :    checksumCalc calculates the checksum for the message in the buffer pucMsgBuf

 Arguments   :    pucMsgBuf – Buffer with the message whose checksum needs to be calculated

                  IMsgLen – Number of characters in the buffer pucMsgBuf

                  pucChkSum – Pointer to an unsigned char in which the calculated checksum will be
                  filled in

 Return type:     void


*****************************************************************************/

 void checksumCalc(unsigned char *pucMsgBuf, int iMsgLen, unsigned char *pucChkSum)

 {

      unsigned char pucTmpChar;

      pucTmpChar = 0;

      *pucChkSum = 0;

      while(iMsgLen --)

      {

           pucTmpChar += * pucMsgBuf;

           pucMsgBuf++;

      }

      pucTmpChar = pucTmpChar % 256;

      * pucChkSum = - pucTmpChar;

      printf("Check sum is %1X\n",* pucChkSum);

 }
```

ASCII Characters

Table 46. ASCII Character Table

Dec	Hex	Char	Dec	Hex	Char	Dec	Hex	Char	Dec	Hex	Char
0	0	NUL	32	20		64	40	@	96	60	`
1	1	SOH	33	21	!	65	41	A	97	61	a
2	2	STX	34	22	“	66	42	B	98	62	b
3	3	ETX	35	23	#	67	43	C	99	63	c
4	4	EOT	36	24	$	68	44	D	100	64	d
5	5	ENQ	37	25	%	69	45	E	101	65	e
6	6	ACK	38	26	&	70	46	F	102	66	f
7	7	BEL	39	27	'	71	47	G	103	67	g
8	8	BS	40	28	(	72	48	H	104	68	h
9	9	TAB	41	29	)	73	49	I	105	69	i
10	A	LF	42	2A	*	74	4A	J	106	6A	j
11	B	VT	43	2B	+	75	4B	K	107	6B	k
12	C	FF	44	2C	,	76	4C	L	108	6C	l
13	D	CR	45	2D	-	77	4D	M	109	6D	m
14	E	SO	46	2E	.	78	4E	N	110	6E	n
15	F	SI	47	2F	/	79	4F	O	111	6F	o
16	10	DLE	48	30	0	80	50	P	112	70	p
17	11	DC1	49	31	1	81	51	Q	113	71	q
18	12	DC2	50	32	2	82	52	R	114	72	r
19	13	DC3	51	33	3	83	53	S	115	73	s
20	14	DC4	52	34	4	84	54	T	116	74	t
21	15	NAK	53	35	5	85	55	U	117	75	u
22	16	SYN	54	36	6	86	56	V	118	76	v
23	17	ETB	55	37	7	87	57	W	119	77	w
24	18	CAN	56	38	8	88	58	X	120	78	x
25	19	EM	57	39	9	89	59	Y	121	79	y
26	1A	SUB	58	3A	:	90	5A	Z	122	7A	z
27	1B	ESC	59	3B	;	91	5B	[	123	7B	{
28	1C	FS	60	3C	<	92	5C	\	124	7C	|
29	1D	GS	61	3D	=	93	5D	]	125	7D	}
30	1E	RS	62	3E	>	94	5E	^	126	7E	~
31	1F	US	63	3F	?	95	5F	_	127	7F	DEL

Appendix B

Level 4 Error Codes

Native Protocol level 4 error codes are listed in Table 47.

Table 47. Level 4 Error Codes for Native Protocols

Decimal Code	Hex Code	Meaning
1	1	Directed Response Error
2	2	Unknown Error Code
3	3	System Error
4	4	System Table Error
5	5	Not Implemented
6	6	Semaphore Create Error
7	7	Semaphore Give Error
8	8	Semaphore Take Error
66	42	Unknown Dest Name
67	43	Unknown Source Name
68	44	Unknown Salvo name
69	45	Bad Level Bit Map
70	46	Invalid Control Level
71	47	Panel Locked
72	48	Chop Lock
73	49	Salvo Lock
74	4A	No Monitor Control
75	4B	Send To MCPU Error
76	4C	Redirect CoProc Msgs Err
77	4D	Assignments Not Enabled
78	4E	New Net Detected, But Not Active
79	4F	Previously Detected Net Now Not Active
128	80	Unknown Command
129	81	CL-CMD Disabled
130	82	Bad CL-CMD Syntax
131	83	Bad Nbr of Sources
132	84	BadError Code

Decimal Code	Hex Code	Meaning
133	85	Parse EOT missing
134	86	Parse HT missing
135	87	Parse Bad Protect Flag
136	88	Parse Bad Dst Name
137	89	Parse Bad Src Name
138	8A	Too Many Sources
139	8B	Bad Parameter
140	8C	Bad Mask
141	8D	Unknown Tag For RCL
142	8E	Chksum Lvl4 Err
143	8F	Lvl4 Embedded SOH Err
144	90	Lvl4 Embedded EOT Err
145	91	Bad Dst Index
146	92	Unknown Dst Index
147	93	Bad Src Index
148	94	Unknown Src Index
149	95	Bad Level Index
150	96	Invalid Ctl Lvl Index
151	97	Level Not In Destination
152	98	Rooms Not Enabled
153	99	Room Count Is Zero
154	9A	No Dest Status Exists
155	9B	Err Trying To Set Time in MCPU
156	9C	Date format error
157	9D	Time format error
158	9E	Parse Bad Salvo Name

Router Control Language level 4 error codes are listed in Table 48. The Description column is the text returned by the RCL Server when queried by the RCL Client. The Condition column is what caused the error.

Table 48. Level 4 Error Codes for Router Control Language Protocol

Decimal Code	Hex Code	Description	Condition
1	1	Error code not used in RCL.	Error code not used in RCL. Maintained for NP compatibility.
2	2	Unknown error code	Error code sent by QE, error_code is not known.
3	3	Internal system error	Catastrophic system error
4	4	Error code not used in RCL.	Error code not used in RCL. Maintained for NP compatibility.
5	5	Command not implemented	The Command sent by the client is not implemented.
6	6	Error code not used in RCL.	Error code not used in RCL. Maintained for NP compatibility.
7	7	Error code not used in RCL.	Error code not used in RCL. Maintained for NP compatibility.
8	8	Error code not used in RCL.	Error code not used in RCL. Maintained for NP compatibility.
66	42	Unknown destination name	Destination name specified in operation not known in the router.
67	43	Unknown source name	Source name specified in operation not known in the router.
68	44	Unknown salvo name	Salvo name specified in operation not known in the router.
69	45	Unknown level bitmap	Level specified in operation (using level bit map) is bad.
70	46	Invalid control level	Level specified in operation is not among the set of controllable levels configured for the device
71	47	No permission. Configured as a query only device	Device has been configured as query only device.
72	48	No permission. Chop not allowed	Chop lock has been configured for the device. Hence cannot perform chop using the specific device.
73	49	No permission. Salvo not allowed	Salvo lock has been configured for the device. Hence cannot perform salvo using the specific device.
74	4A	No permission. Monitor control not allowed	Monitor lock has been configured for the device. Hence cannot perform monitor using the specific device.
75	4B	Error code not used in RCL.	Error code not used in RCL. Maintained for NP compatibility.
76	4C	Error code not used in RCL.	Error code not used in RCL. Maintained for NP compatibility.
77	4D	Error code not used in RCL.	Error code not used in RCL. Maintained for NP compatibility.
78	4E	Error code not used in RCL.	Error code not used in RCL. Maintained for NP compatibility.
79	4F	Error code not used in RCL.	Error code not used in RCL. Maintained for NP compatibility.
128	80	Unknown command	Command sent is not known.
129	81	Error code not used in RCL.	Error code not used in RCL. Maintained for NP compatibility.
130	82	Error code not used in RCL.	Error code not used in RCL. Maintained for NP compatibility.
131	83	Invalid number of sources	Number of sources specified in `TA` and `TJ` not known
132	84	Invalid error code	Error code sent as part of `QE` is bad.
133	85	Incomplete command packet	`EOT` is not present in the packet.
134	86	Separator field missing	`HT` is not present in the packet.
135	87	Invalid protect flag	Not used.
136	88	Invalid destination name	Destination name specified in operation is bad.
137	89	Invalid source name	Source name specified in operation is bad.
138	8A	Too many sources specified in Take command	Too many sources specified in Take command
139	8B	Invalid parameter	Parameter sent for `BK` command is bad

Table 48. Level 4 Error Codes for Router Control Language Protocol - (continued)

Decimal Code	Hex Code	Description	Condition
140	8C	Invalid flags	Mask sent for `BK, f` command is bad
141	8D	Invalid client identifier	Invalid client identifier
142	8E	Checksum error	Check sum sent in the packet is bad
143	8F	Error code not used in RCL.	Error code not used in RCL. Maintained for NP compatibility.
144	90	Error code not used in RCL.	Error code not used in RCL. Maintained for NP compatibility.
145	91	Invalid destination index	Destination Index specified in operation is bad.
146	92	Unknown destination index	Destination name specified in operation is not known in the operation
147	93	Invalid source index	Source Index specified in operation is bad.
148	94	Unknown source index	Source name specified in operation is not known in the operation
149	95	Invalid level Index	Level Index specified in operation is bad.
150	96	Invalid control level Index	Level specified in operation is not among the set of controllable levels configured for the device
151	97	Destination is not present in the level	Destination is not present in the level specified in operation
152	98	Error code not used in RCL.	Error code not used in RCL. Maintained for NP compatibility.
153	99	Error code not used in RCL.	Error code not used in RCL. Maintained for NP compatibility.
154	9A	Unknown destination status	Destination specified in the operation does not have any status
155	9B	Error setting time	Error trying to set time using `ST`
156	9C	Date format error	Date format error while trying to set time using `ST`
157	9D	Time format error	Time format error while trying to set time using `ST`
158	9E	Invalid salvo name	Salvo name specified in operation is bad.
159	9F	Router not available	Router not available in the specified area.
160	A0	No router present	No router present in the network to perform operations on.
161	A1	Invalid area index	Area index specified as part of the operation is invalid.
162	A2	Invalid area bitmap	Area bitmap specified as part of the operation is invalid.
163	A3	Invalid area name	Area name specified as part of the operation is invalid.
164	A4	No level present in destination	The destination specified is not present in any level.
165	A5	No level present in source	The source specified is not present in any level.
166	A6	Level(s) Excluded	The level(s) specified are excluded with respect to this client.
167	A7	Destination protected	The destination specified is protected and hence the operation cannot be carried on this destination.
168	A8	RCL connection not established	The requested operation cannot be carried out before establishing RCL connection.
169	A9	Time out error	Timed out error in sending the response for the requested operation.
170	AA	No cross point	The requested take cannot be performed, as there is no cross point to establish connection.
171	AB	Invalid subscription type	The subscription type specified in subscribe or unsubscribe request is invalid.
172	AC	Subscription service not initialized	The subscription service was not initialized in the system.
173	AD	Duplicate subscription	The client has already subscribed for the same event.

Table 48. Level 4 Error Codes for Router Control Language Protocol - (continued)

Decimal Code	Hex Code	Description	Condition
174	AE	Subscription Services Shutdown	The subscription Service has been shutdown due to internal errors.
175	AF	Subscription Services reset.	The RCL server has reset the subscription service. All existing subscriptions are lost and the client has to issue all subscription requests again.
176	B0	Time Stamp error.	There is an error in the time stamp provided as part of the operation request.
177	B1	No VITC	Vertical Interval Time Code is not connected to the system.
178	B2	Time out of bound	The time at which the operation has been requested is out of bounds.

Index

A

B

C

RCL = Router Control Language, NP = Native Protocol, SNC = Serial Node Controller

D

E

F

G

H

I

L

M

N

O

P

Q

R

S

T

U

W