GRASS VALLEY GROUP

SERIES 7000
ROUTERS

SIGNAL MANAGEMENT SYSTEM

Protocol Manual
071-0201-01

Publication Date: 11-30-99

Contacting Grass Valley Group

		Voice	Fax	Address	Web Site
Customer Support	North America	(800) 547-8949 (530) 478-4148	(530) 478-3181	Grass Valley Group P.O. Box 1114 Grass Valley, CA 95945, USA	www.grassvalleygroup.com
	Elsewhere	Distributor or sales office from which equipment was purchased.			
Product, Service, Sales Information	North America	(800) 547-8949	(503) 627-7275	P.O. Box 500 M.S. 58-965 Beaverton, OR 97077-0001, USA	www.grassvalleygroup.com
	Europe	44 (10) 1628 40 3300	44 (0)1628 40 3301		
	Asia	(852) 2585-6688	(852) 2802-2996		
	Japan	81 (3) 5992 0621	81 (3) 5992 9377		
	Latin America	(305) 477-5488	(305) 477-5385		

Contents

Section 1

Protocols Overview

Introduction

The Series 7000 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 Series 7000 system:

- Native Protocol (via RS-232,RS-422, SLIP, or Ethernet)
- Pro-Bel/Omnibus 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:

- SystemDiagnostic Interface (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

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

- Networked System 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.

System 7000 equirements differ for different control mechanisms, and are described where appropriate.

Refer to the manufacturer's documentation of any external device being interfaced to a System 7000 for information on installation and configuration procedure for these systems

Section 2

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 in this section are shown in Table 2-1.

Table 2-1. Notation symbols

Symbol	Meaning
...	A continued sequence.
|	Or
[ ]	Optional parameters
< >	Brackets choices, ASCII control characters, and 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 communications 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 communications rates.

Pinouts and cable diagrams are available in the *Series 7000 Installation Manual*.

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 *Series 7000 Configuration Manual* 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 2-2:

Table 2-2. 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 (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. **ACK**s are returned immediately, with no field delay. **ACK**s 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. **NAK**s are returned immediately, with no field delay. **NAK**s 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 **NAK**s 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 2-18* 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 2-17* 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 2-21* 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

Standard TCP/IP stream sockets (sometimes referred to as Berkeley sockets) are used to connect to and communicate with a Series 7000 MCPU. Stream sockets are connection oriented, a connection must be opened and maintained for the duration of 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 2-3 may be used:

Table 2-3. 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 Appendix A in the *Configuration Manual* 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 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 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 2-17* 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 comments 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 2-4.

Table 2-4. 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 2-5. 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 2-6.

Table 2-6. 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 upper case letters are used in a in a name, NODE_CON for example, the system will 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

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 **NAK**ing 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 Hex number 71 - 79 (Table 2-7).

Table 2-7. Level 2 NAK Error Codes

Hex Value	Meaning	Description
71	Buffer Size Exceeded	The number of characters received since the last detected **SOH** is greater than the maximum Native Protocol message length.
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.
73	Undefined	Undefined
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.
75	Checksum Error	Packet had a bad checksum. Low Level errors such as framing, overrun, etc., are reported as checksum errors.
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).
77	Missing SOH	No SOH detected in message. Results when EOT is detected without a preceding SOH.
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).
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 re-send 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.

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, see the *Series 7000 Service Manual*.

If the system is reporting Level 4 error messages resulting from take commands, check the **Cfgd NativeProtocol 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 2-13*. 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 2-8).

Table 2-8. 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 2-9.

Table 2-9. 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 Destina-tion 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 Des-tination	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 2-10. 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.

Table 2-10. BK Command Parameters - (continued)

Parameter	Description
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.
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 2-11:

Table 2-11. KB Response Parameters

Parameter	Description
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 above).
E	Echo = ON | OFF. A value will be returned for both set and query requests (see above).
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 below for more information.

KB, F Response

For a **`KB, F`** response, the **`data`** is defined in Table 2-12:

Table 2-12. 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.

Note that **`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'>,src_name,level_bitmap,[device_name],
[chop_src_name]
```

Parameters for this response are explained in Table 2-13.

Table 2-13. 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 **NAK**s) 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 2-14.

Table 2-14. 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 2-15.

Table 2-15. 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 2-16.

Table 2-16. 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

Responses

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`

of EndLevels IN THIS ENTRY, & 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,S,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 2-17).

Table 2-17. 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 3

Serial Node Controller Protocol

Introduction

This section describes the format of the Series 7000 Serial Node Controller 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.
- 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,
- 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, 2's compliment. Includes only <command> and <message> fields. Does not include <SOM>. Checksum MSB = 0.

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, 2's compliment of message field. Does not include SOMx, COUNT, or BIT7ESCs. Calculated before BIT7ESCs are placed in transmit packets and after packet normalization in receive. Checksum MSB = 0.

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. e.g, 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

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).

Interrogate (1)

Direction:	Controlling Device -> Series 7000 Node Controller
Purpose:	Request for Destination Status.
Rate:	On demand.

Table 3-1. 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 3-2. 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`** command.

Rate: On demand.

Table 3-3. 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.

Rate: On demand.

Table 3-4. 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 3-5. 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 3-6. 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`.

Rate: On demand.

Table 3-7. Connect_On_Go_Acknowledge Command

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

Rate: On demand.

Table 3-8. 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 - ASCII Characters

Figure A-1. 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

Figure A-1. ASCII Character Table

Dec	Hex	Char	Dec	Hex	Char	Dec	Hex	Char	Dec	Hex	Char
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

Index