Section 1: Port Configuration

The 7200A can be operated remotely from an instrument controller or computer across the General Purpose Interface Bus (GPIB (with option IF-1)) , or from a computer or terminal with an RS-232-C serial connection. Commands sent over either interface can set or read any 7200A instrument mainframe or plug-in front panel function. However, only one port can be enabled at a time for controlling the 7200A remotely.

Communication to the 7200A through either the GPIB or the RS-232-C interface is based on industry standard protocols and is as similar as the two standards allow. An introduction and setup description of each interface is given, followed by a description of how to control the 7200A from either interface.

Selecting the Computer Port

The 7200A provides three interfaces: the General Purpose Interface Bus (GPIB) (optional), an RS-232-C Serial Port, and a Centronics Port. The GPIB and RS-232-C ports can be used to connect the 7200A to a host computer or to connect to a printer or plotter for hardcopy output. The Centronics port is used for hardcopy output only. The advantage of GPIB is high transmission rates. The advantage of RS-232-C is that it is a lower cost interface which can generally be used over longer distances. If the 7200A will be used in conjunction with a host computer, set the "Remote Control from" field in the Communications screen to either GPIB or RS232.

GPIB Remote Control

The General Purpose Interface Bus (GPIB) is originally based on the IEEE Standard 488, 1976 (and later revised by IEEE 488.1,1987).

The GPIB can interconnect many instruments to allow communication with one another over shared cables. The GPIB uses a bit-parallel, byte-serial format. The 7200A can achieve a maximum transmission rate of 400 kBytes per second.

A device connected to the GPIB is either a talker, listener, or controller. Although some devices can change roles, a device can perform just one role at a time.

Talker — Places messages or data on the network for transmission to other devices. Only one device on the network can be the talker.

Listener — Receives data or commands over the network. Several listeners may be active at one time.

Controller — Governs the operation of the network. A controller, usually a computer, normally sends program messages to devices and receives response messages from them. One controller task is to decide which device is the talker and which is a listener(s). The controller may assign itself to be the talker at one time, and a listener at other times. If devices on the network never change their roles, a controller is not required.

The Communications Screen allows you to select GPIB as the Remote Control port and set the GPIB address for the 7200A. The Hardcopy screen allows you to select GPIB as the hardcopy port for printers and plotters. If GPIB is the selected port for hardcopy, no controller is needed and all other devices on the bus must be in "Listen Only" mode.

GPIB Signals and Lines

The GPIB has 16 signal lines and eight ground return lines. Eight of the 16 signal lines form a bi-directional data bus which transfers data and commands. The remaining eight signal lines control the bus operation. Three lines are for handshake signals which synchronize data transmission. The remaining five are management lines which control the flow of information across the interface.

Setting the GPIB Address

The GPIB address is set in the Communications screen. From the Main Screen, press the Configure System softkey to display the Configure System setup screen. Then press the Communication Setup softkey to display the Communications Setup screen. Move the box onto the "Remote Control from" field and select GPIB. Then move the box onto the "GPIB Address" field and select an address from 0 to 30.

GPIB Host and Hardcopy Operation

The 7200A can communicate across GPIB as a talker or a listener with a remote host controller to receive remote commands/queries and send responses. For this talker/listener remote control operation, the 7200A conforms to the guidelines specified by IEEE 488.2. The hardcopy output can also communicate across GPIB in one of two ways. First, if the hardcopy port is the same as the remote control port, then a remote hardcopy command sends the output to the remote host as a query response. Second, if the hardcopy port is different from the remote control port or and the local hardcopy key is pressed, then the 7200A enters Talk Only mode and does not expect any controller present on the bus.

Remote Control Operation over GPIB

Talk/Listen

The 7200A enters this mode when the "Remote Control from" field in the Communications Setup screen is set to GPIB. In this mode, the 7200A can both receive commands and setups from the remote host computer and send data and measurement results.

Hardcopy Operation over GPIB

Talk Only

To output hardcopy data over GPIB, the "Hardcopy Port" field in the Hardcopy screen must be set to GPIB. Setting the Hardcopy Port has no effect on the selected port until the hardcopy is initiated. If the Hardcopy Port is GPIB, then pressing the local Hardcopy key will force the 7200A to enter Talk Only mode. Also, if the Hardcopy Port is GPIB and the Remote Control port is RS-232-C, then initiating a hardcopy remotely from RS-232-C will also force the 7200A to enter Talk Only mode. Talk Only is a special GPIB mode where there is no controller allowed on the bus; the 7200A is the only talker and all connected devices must be listeners (ie., printers/plotters must be in Listen Only mode). However, if both the hardcopy port and "Remote Control from" field are set to GPIB, then pressing the local Hardcopy key

just sets the User Request (URQ) bit in the Standard Event Status (*ESR) register. the 7200A cannot enter Talk Only mode since this may disrupt the controller. Instead, the controller may query the *ESR register and if the URQ bit is set, the controller may halt bus activity and synchronously initiate a remote Hardcopy as describes next.

Talk/Listen

When both the Hardcopy Port and the Remote Control port are set to GPIB, then sending the remote command "HARDCOPY" or "HCPY" over GPIB from the host computer will cause the 7200A to send the hardcopy output to the host computer as a response message. In this mode, the 7200A will wait to be addressed to talk before sending the hardcopy data. The host computer then has three options in generating the hardcopy:

1) The host computer may read the data into internal memory and then send the data to a printer/plotter. This is exactly the same as reading a query response.

2) The host computer may send the "HARDCOPY" remote command and then address the printer/plotter to listen and the 7200A to talk and read the data from the 7200A. As the data is read into the computer's internal memory, it is also printed/plotted to the printer/plotter which is a Listener.

3) The host computer may send the "HARDCOPY" remote command and then address the printer/plotter to listen, the 7200A to talk, and the controller to go into stand-by mode waiting for EOI. Alternatively, the Data Processing Status Register (DPR) could be programmed to issue an SRQ when hardcopy is complete so that the host computer can perform other tasks while the hardcopy is performed.

Hardcopy port	Remote Control Port	Action of Local Hardcopy Key	Action of Remote HCPY command
GPIB	GPIB	Sets the URQ bit in the *ESR register.	Hardcopy data output when the controller addresses the 7200A to talk.
GPIB	RS232	Hardcopy data output in Talk-Only mode. Address the device at address 30 to listen before sending data.	Hardcopy data output in Talk-Only mode. Address the device at address 30 to listen before sending data.
RS232	RS232	Sets the URQ bit in the *ESR register.	Hardcopy data output when controller asserts CTS (Hardwire mode) or until receipt of XOFF.
RS232	GPIB	Hardcopy data is output immediately.	Hardcopy data is output immediately.
Centronics	GPIB/ RS232	Hardcopy data is output immediately.	Hardcopy data is output immediately.
Floppy	RS232/ GPIB	Hardcopy data is written to a floppy disk immediately.	Hardcopy data is written to a floppy disk immediately.

GPIB Device Interconnections

The devices on the GPIB network may be connected in any combination of star or linear arrangements (Figure 1.1). Standard IEEE 488.2 cables must be used to connect all the devices and total length must not exceed 20 meters. The devices must conform to these rules:

- At least half the devices on the network must be turned on.
- One network can connect no more than 15 devices (including the controller).
- One device must be connected for every two meters of cable, assuming one device presents one standard device load. The 7200A's GPIB connector is located on its rear panel.
- Each device must have a unique bus address.

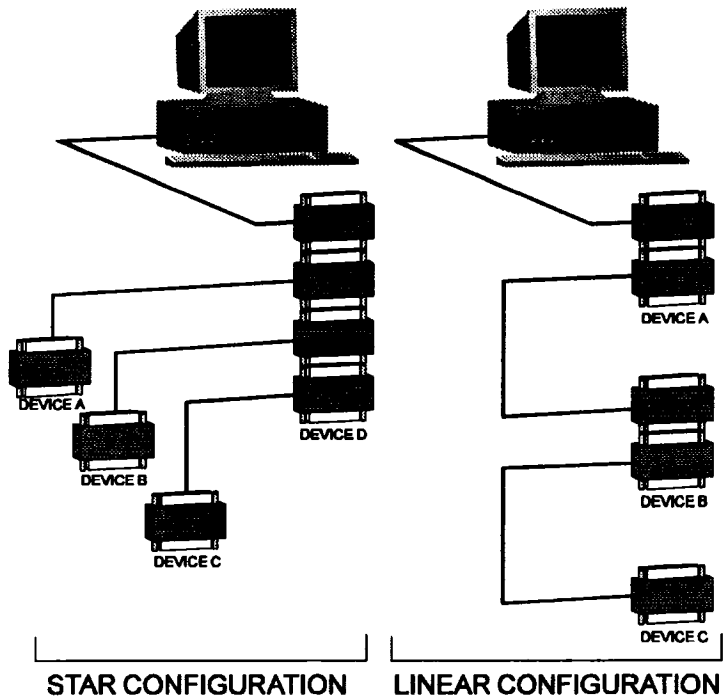


Figure 1.1: Examples of GPIB Network Arrangements

The 7200A interface is defined by the following GPIB function codes:

For a description of these functions and their subsets, see IEEE Standard 488.1, Section 2.2 through 2.12.5. The IEEE Standard is published by the Institute of Electrical and Electronics Engineers, Inc., 345 East 47th Street, New York, New York 10017.

INTERFACE FUNCTION	COMMENTS
Controller (C0)	No controller capability.
Source Handshake (SH1)	Complete source handshake capability.
Acceptor Handshake (AH1)	Complete acceptor handshake capability
Talker (T6)	Basic talker with serial poll capability and unaddress* if MLA (My Listen Address).
Listener (L4)	Basic listener with unaddress if MTA (My Talk Address).
Service Request (SR1)	Complete serial poll capability.
Device Trigger (DT1)	Capable of responding to device trigger.
Device Clear (DC1)	Responds to device clear (universal or selective).
Parallel Poll (PP0)	No parallel poll capability
Remote Local (RL1)	Complete remote/local capability
Electrical Interface (E2)	SRQ, NRFD, and NDAC are tri-state lines

*Unaddress refers to the action taken when the interface switches its function. This action effectively clears the current function before the next function is selected.

Table 1.1: 7200 IEEE-488 Function Codes

RS-232-C Remote Control

The 7200A can also be operated from a computer or terminal via its RS-232-C port. RS-232-C uses serial transmission and complies with the Electronic Industries Association's RS-232-C standard. (The equivalent international standard is ISO V24 which is generally compatible with the RS-232-C version.)

Unlike the GPIB where many devices can be interconnected, the RS-232-C connects just two devices. Only three communication lines are necessary to establish the interface: transmitted data, received data, and logic ground. However, the additional lines, RTS (request to send) and CTS (clear to send), permit transfer of data only after confirming that the receiving

device is capable of accepting more data. That is, the sender sends an RTS and waits for a CTS from the receiver before sending data. This protocol guarantees that data does not overrun the receiver's buffer.

RS-232-C offers compatability with most computers. It uses a bit-serial data format with a maximum transmission rate of 19,200 bits per second, much less than that of GPIB.

Each data word is transmitted as a separate packet with its own start and stop markers, or bits. The RS-232-C standard defines the electrical characteristics of these bits and the composition of each packet. Their composition and transmission rate must be the same for both the device and the 7200A. The Communications Setup screen is used to select transmission rate, error checking (parity), and number of stop bits. In order to establish communications, additional serial transmission characteristics may be set remotely using the COMM_RS232 remote command. See Section 5: Communication Commands for a description of this command.

RS-232-C Configuration

Setup the Serial Port

The 7200A contains a 9-pin, male RS-232-C connector for serial communication with a printer, terminal, or computer. To connect an RS-232-C line to the 7200A, use a female DB9-D connector. If the computer has a DB25-D connector, use a DB9-D to DB25-D cable adapter. The optional CTS and RTS handshaking guarantees that data passed between a remote computer and the 7200A will not overrun the 7200A or the computer's RS-232-C buffer.

Select the desired settings for the interface using the Communications Setup screen:

1. From the Main Screen, press the Configure System softkey to display the Configure System setup screen.
2. Then press the Communication Setup softkey to display the Communications Setup screen.

RS-232-C Host Interconnection

Although the RS-232-C standard defines signal lines and electrical characteristics, it does not define mechanical characteristics. The 7200A RS-232-C output port is configured as an RS-232-C Data Terminal Equipment so that data is sent from pin 2 and received on pin 3. For remote operation, the RS-232-C port must be connected to a computer terminal. The following diagrams are used for various host drivers.

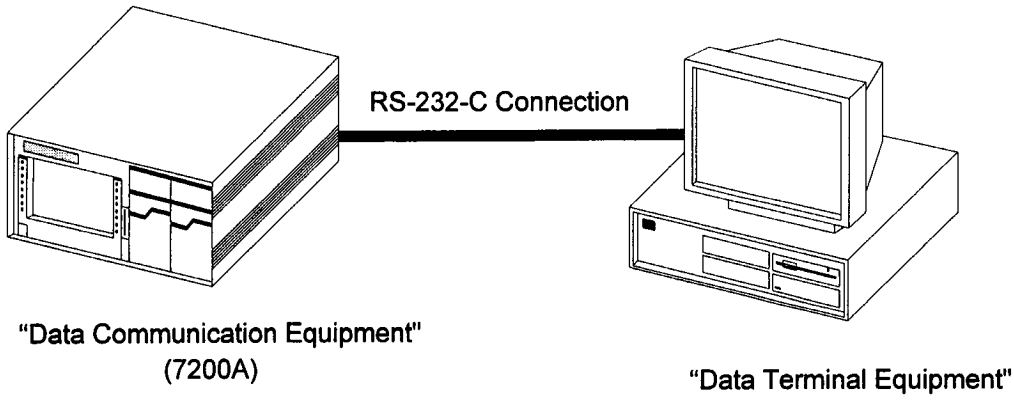


Figure 1.2: RS-232-C Connection to an IBM-PC Host

DB9 to DB25 Wiring

This wiring configuration is used for IBM-PCs and compatibles with DB25-D connectors configured as Data Terminal Equipment. Note that for XON-XOFF communication protocol, only pins 2, 3, and 5 on the DB9-D connector are needed. Also, commercially available DB9-to-DB25 adapter cables for the IBM-PC swap pins 2 and 3 and pins 7 and 8.

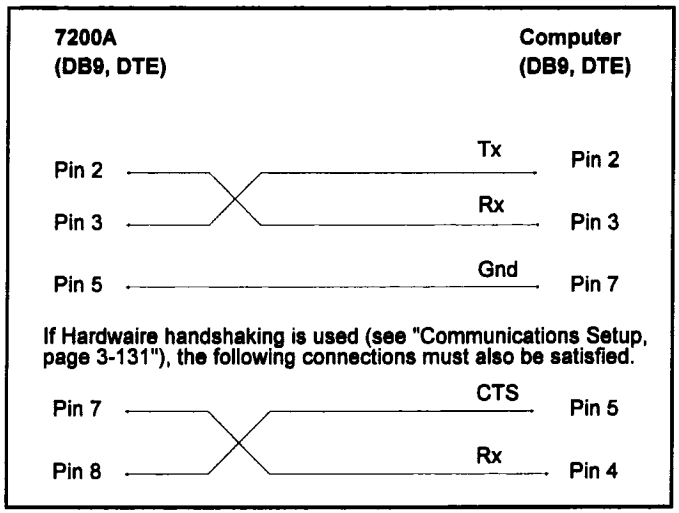
7200A (DB9, DTE)		**Computer (DB25, DTE)**
Pin 2	Tx	Pin 2
Pin 3	Rx	Pin 3
Pin 5	Gnd	Pin 7

If Hardwaire handshaking is used (see "Communications Setup, page 3-119"), the following connections must also be satisfied.

Pin 7	CTS	Pin 5
Pin 8	RTS	Pin 4

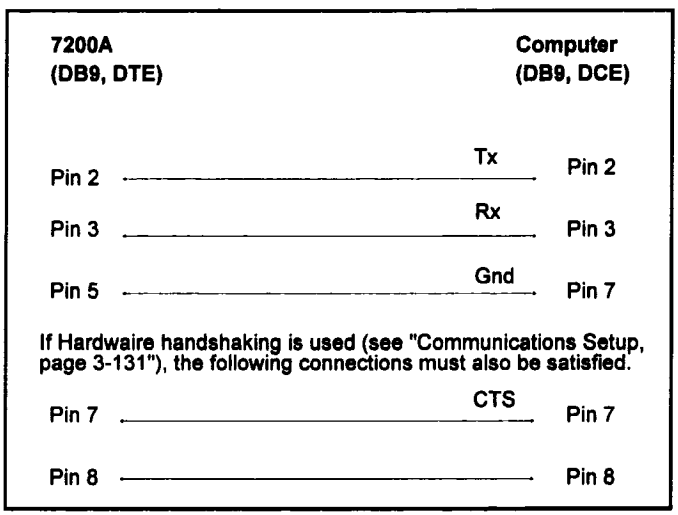
DB9 to DB9 Wiring

For IBM PC-AT types with DB9-D connectors configured as Data Terminal Equipment.



DTE to DCE Wiring

For non-IBM types with DB9-D connectors configured as Data Communications Equipment.



RS-232-C Interconnections for Hardcopy

When connecting an RS-232-C serial printer/plotter to the 7200A, the printer/plotter configuration must match the 7200A RS-232-C port settings. To modify settings, use the Communications Setup screen.

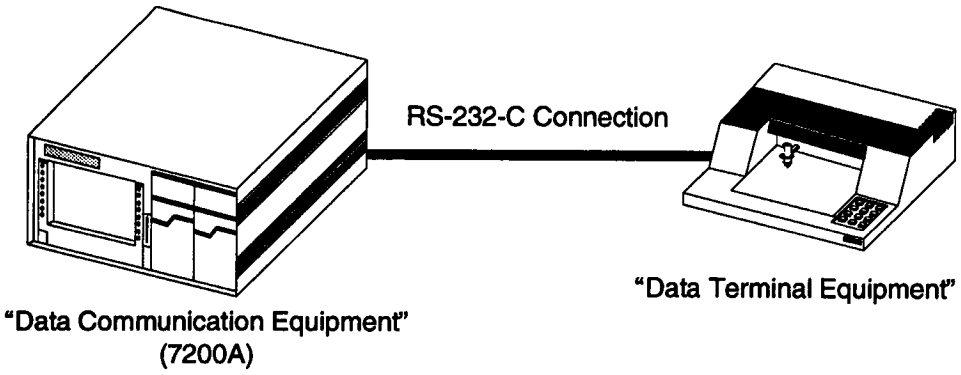
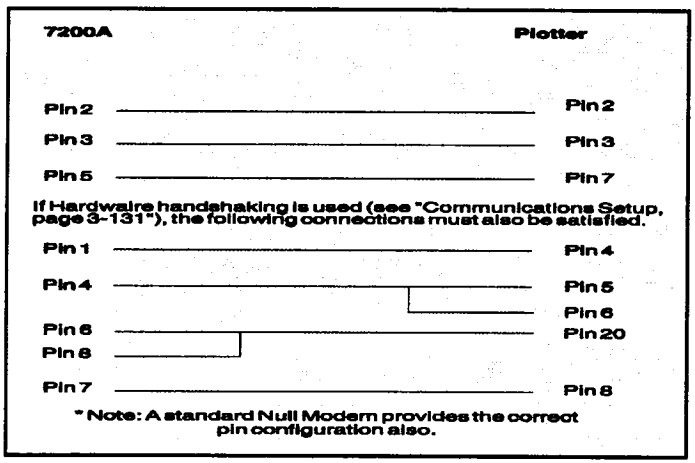


Figure 1.3: RS-232-C Connection to an RS-232-C Serial Plotter

DB9 to DB25 Wiring

NOTE: The 7200A RS-232-C interface is a DB9-D connector. Use an adapter cable to connect to an RS-232-C DB25-D connector.



Parallel-Centronics Wiring

The 7200A uses a standard DB25-D female connector as the Centronics parallel output port. An adaptor cable is required to adapt the 7200A DB25-D connector to the standard 36-pin bail lock connector used on most Centronics printers.

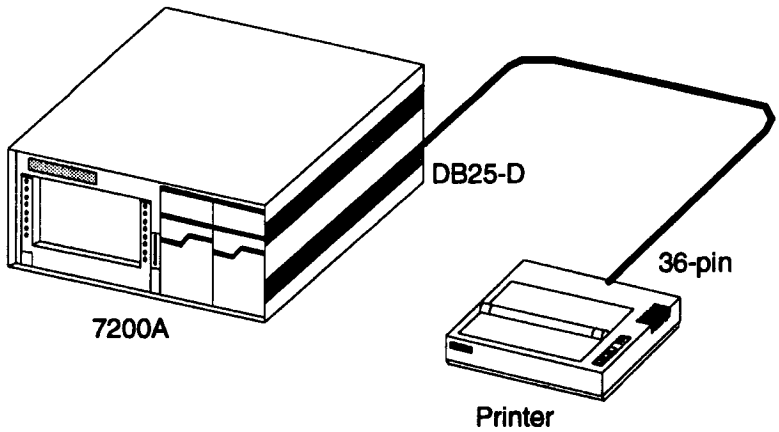


Figure 1.4: Output to Centronics type Printer/Plotter

RS-232-C Host Operation

The 7200A may be controlled by a Remote Host computer in a similar manner as in GPIB. It is able to accept commands, strings, and arbitrary block data and send back responses to queries. However, RS-232-C communications is limited to the transfer of ASCII characters in the range 1 to 127. Also, any character whose value is below a <space> (ASCII 32) can not be used as part of a valid command or query but may be used as a valid <PROGRAM MESSAGE TERMINATOR>. The exception to this rule is the <ESCAPE> character (ASCII 27). When <ESC> is sent to the 7200A, the very next character sent is interpreted to have a special meaning.

The valid Escape sequences are as follows:

Command	Description
<ESC>(	Selects HARDWIRE handshake mode
<ESC>)	Selects XON-XOFF handshake mode
<ESC>[	Selects Echo off (half-duplex mode)
<ESC>]	Selects Echo on (full-duplex mode)
<ESC>C	Sends a DCL (device clear) command
<ESC>R	Sends a REN (remote enable) command
<ESC>L	Sends an LCL (local enable) command
<ESC>F	Sends an LLO (local lockout) command
<ESC>T	Sends a GET (group execute trigger) command

Table 1.2: Valid Escape Sequences

All <ESC> commands are immediately executed upon being parsed. Their intent is to simulate GPIB commands over the serial port.

When the 7200A receives ASCII block data in excess of its input buffer size it will send XOFF (ASCII 19) to hold up the transfer of data from the Remote Host until it has processed the current buffer. Also, if the handshake mode is HARDWIRE, it will de-assert CTS (Clear To Send). When the 7200A is ready for more data, it will send XON (ASCII 17) and assert CTS.

For a complete description of setting the configurtation of the RS-232-C port for Remote Host communications, see the COM_RS232 remote command in Section 5 (Communication Commands).

Section 2: Command Syntax

The following segments describe the rules and syntax for controlling the 7200A from a remote computer over either GPIB or RS-232-C. Any differences between the ports are noted.

Message Types

Commands

Commands have two categories: action and query.

An action command causes the 7200A to make an assignment or perform a function. For example, it might cause the 7200A to calibrate all the plug-ins, or an assignment may result in a new front panel setting, a communication parameter receiving a new value, or the date being set.

Commands that request results are called queries. They ask the 7200A to return waveform data, settings, or measurements.

Responses

These are replies sent from the 7200A in response to query commands.

Waveforms

Waveform data is a special form of response. It may be output in binary or hexadecimal formatted blocks. These formats are more compact than that used for response messages, so a large number of data points can be transferred in less time.

Status

A status message indicates the 7200A's current internal state.

With few exceptions, all commands, responses, and status messages are encoded according to the American Standard Code for Information Interchange (ASCII) and are strings of printable characters. Upper and lower case characters are interchangeable.

Message Direction

As shown in Figure 2.1, the controller sends commands to the 7200A, and the 7200A sends waveforms, responses, and status messages back to the controller.

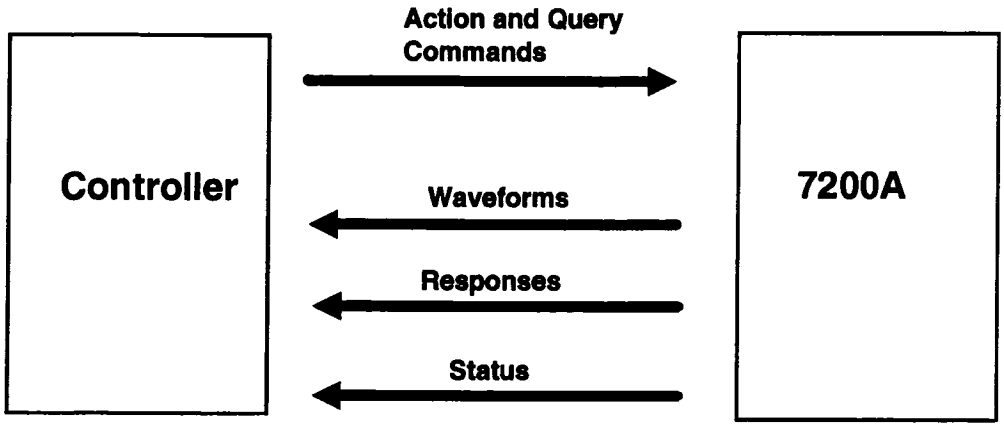


Figure 2.1: Message Directions

Command Processing

Commands are not processed until the 7200A receives an **<end>**, or, in the case of waveform input, when the 7200A input buffer is full (i.e., normally, no action is taken on any part of a command message until the entire message is received or the message size exceeds the input buffer size).

Command Processing Order

Valid commands are processed in the order they are received.

Some remote commands cannot be performed immediately. For example, it is not possible to read channels that are armed and waiting for a trigger, since the memories associated with these channels are continuously being written. If the 7200A receives a command it cannot perform immediately (another example is the STORE of a channel), it defers executing the command until the needed waveform is acquired.

Command Errors

Before attempting to execute a command or query, the 7200A confirms that all the required parts of the command are provided, and that all the arguments are within required ranges.

If an error is generated, the 7200A will set the appropriate status and, if enabled, report it to the host computer. The host can then interrogate the status byte(s) to determine the nature of the error. Refer to Section 4 for details on status bytes.

NOTE: Commands preceding and following an error in multi-command messages are still executed. This provides consistent operation whether commands are sent one at a time or several per message.

Output from the 7200A

When the 7200A generates a response to a query, the controller should read it before sending a second query. If the controller sends a second query before reading the response to the first one, the 7200A interprets this as an Interrupted Action and performs the following:

1. Upon receiving the **<end>** of the second query, the 7200A flushes its output buffer of all responses to previous queries.
2. The 7200A sets a Query Error bit, and
3. The 7200A fills the output buffer with the response to the second query.

IEEE-488 Standard Messages

This section explains how the 7200A reacts to the Standard 488.2 messages.

NOTE: This section pertains to GPIB only

Serial Poll Function

The 7200A implements a full Serial Poll Interface Function:

1. It can assert the SRQ (Service Request) control line.

2. It will respond with the current serial poll byte or STB when addressed to Talk and after the Serial Poll Enable interface message is received.

3. After transmitting its status message, the 7200A stops asserting the SRQ line and clears its internal status byte.

Receiving the Trigger Message

The 7200A responds to the Trigger message [Group Execute Trigger (GET) or the *TRG command] by arming all plug-ins. The trigger signal, also available on the rear panel, can be used

for external hardware or event synchronization to internal 7200A operations. It is executed after all previously received commands have been processed.

Interface Clear

The Interface Clear message (asserting IFC line) is an asynchronous control line that causes all bus activity to halt. When the 7200A receives the IFC message, it becomes unaddressed, stops talking or listening, and will not participate in future bus transactions until readdressed to talk or listen.

Device Clear (Selective or Universal)

The 7200A will respond to a Selective Device Clear or a Universal Device Clear interface message. The former requires that the 7200A first be addressed to listen, followed by the Selective Device Clear message. The latter does not require that the instrument be previously addressed to listen. Device Clear causes the input buffer, the output queue, and the message available (MAV) status bit to be cleared.

Go to Local, Go to Remote, Go to Remote with Lockout Local

The 7200A can operate in Local or Remote mode. In Local mode, all front panel controls are operational and commands from the host computer will also be processed. In Remote mode, the 7200A operates under computer control and no front panel controls are operational except the Local softkey (if enabled). (The 7200A always powers on in Local mode.)

NOTE:The 7200A processes all messages regardless of being in Remote or Local modes.

The 7200A switches to Remote mode (with Local softkey enabled) when the 7200A receives the command "REM", or a command is sent with the REN line asserted. All instrument settings remain unchanged during local-to-remote transitions. The lower left part of the 7200A screen indicates that Remote mode is enabled and the Local softkey appears. No other front panel controls operate.

If the 7200A is under remote control and the Local softkey is pressed, the instrument interrupts program control and returns to local control. Data and/or settings can now be changed locally.

CAUTION: *To prevent a transition back to local mode the 7200A can be placed in a Local Lockout state using the "LLOK" command. In Local Lockout state, all front panel keys and knobs are*

disabled. Once Remote with Local Lockout is set, it can only be cleared when the 7200A is put into Local mode by sending the "LOC" command or readdressing the 7200A with REN deasserted.

Message Syntax

Messages consist of one or more data bytes which are sent over the bus.
All messages sent to and received from the 7200A are formed of English words except for waveform transfers. Abbreviations, typically two to four characters, are also defined to achieve higher throughput. Lower or upper case alphabetic characters are interchangeable.

NOTE: Any message received by the 7200A must conform to IEEE-488.2 syntactic requirements. (If a violation is detected, the 7200A will generate an error which indicates an invalid command.) The syntax for each type of message is described below.

Action Command Syntax

Commands are sent to the 7200A to initiate various actions. They contain Headers, sometimes an Argument(s), and a Terminator:

Header — Identifies what action to take; e.g., set the date, stop acquisition.

Argument(s) — Qualifies or supplements the header. The argument acts as a parameter(s) or data to the header. It is included in the command only if a header is defined to require an argument(s). For example, an argument indicates what date to set.

Terminator — Indicates the end of the command message. GPIB and RS-232-C have different message terminators. In this manual, the command message terminator is represented by **<end>**.

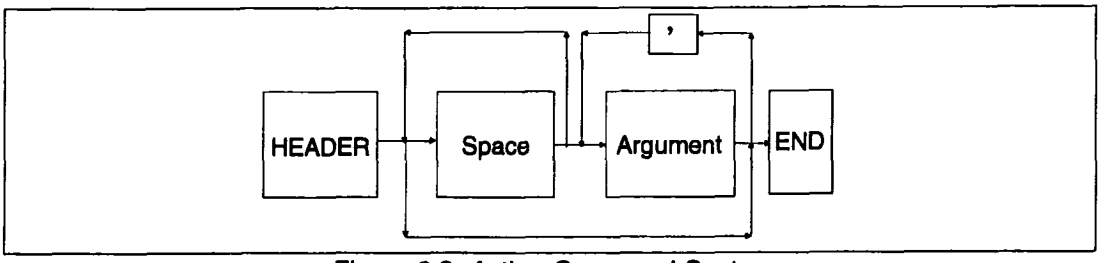


Figure 2.3: Action Command Syntax

Unless specifically noted, white spaces (ASCII 32 decimal) between the parts of a command do not affect its processing. Upper and lower case characters are interchangeable. The general format of a command follows:

Command Header

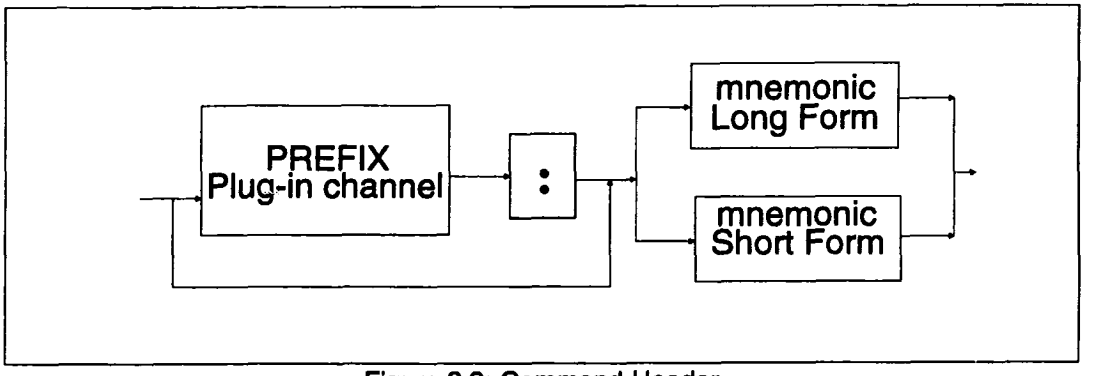


Figure 2.2: Command Header

Command Arguments

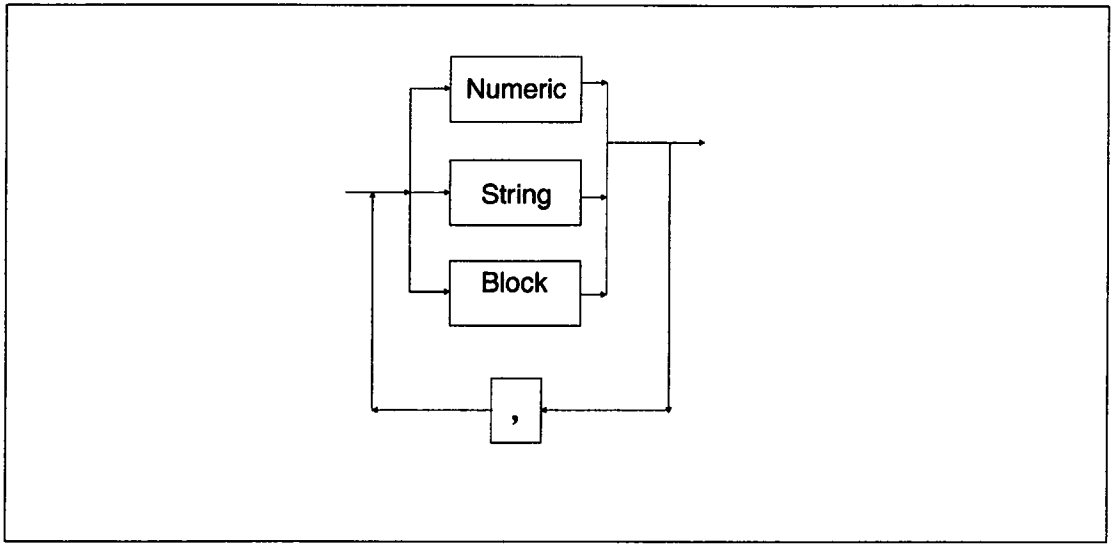


Figure 2.4: Command Arguments

Query Syntax

The Syntax of a Query is very similar to that of an Action command. A Query command adds a question mark ("?") immediately after the last character of the header. For example, to find the current value for the offset on channel 2 of plugin B, send: **B2:OFFSET?**.

NOTE: Many action commands have a corresponding query command which may have arguments.

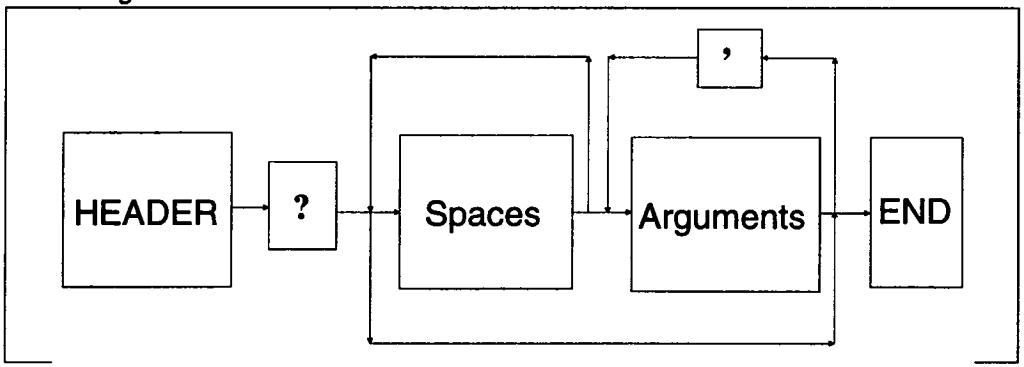


Figure 2.5: Query Syntax

Multiple Commands

A message containing more than one command before the terminator is called a compound, or multiple command message.

Sending a multiple command increases throughput. Each command (header and any arguments) is separated from the following one by a semicolon (";"). Space(s) on either side of the semicolon do not affect processing. Upper and lower case characters are interchangeable.

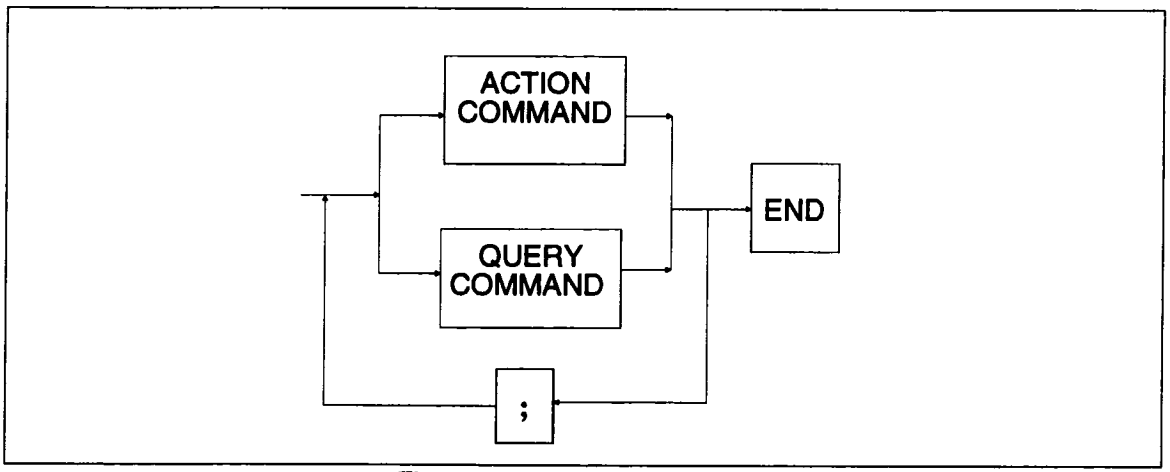


Figure 2.6: Multiple Commands

A multiple command can include Action and Query commands. For example, one multiple command can perform auto setup, request the current time and date being used, and execute a trigger command as follows:

ASET; DATE? ; *TRG <end>

Command Header

A command header defines the action to perform. The header begins with a letter and can be followed by any combination of up to 15 letters, numbers, and underscores. Any command with more than four characters has a short form. (Long and Short form headers can be intermixed.) Using the short form (four or less characters) increases throughput. The long form, however, makes understanding program code easier. For example, to set the timebase remotely, either **TIME_DIV** or **TDIV** can be sent.

Command headers comprise three broad categories according to their syntactic make-up:

- Directed Header,
- System Header, and
- Standard Header.

Directed Header

This type of header directs an action at an object. The object can be either a plug-in, channel, trigger source, or trace. The prefix identifies the object being acted upon. It is followed by a colon (":") and the header which indicates the action performed.

prefix:header

Only one prefix is permitted per header.

NOTE: If a command is defined as having a prefix, the prefix must always be specified.

The types of prefixes are:

Plug-in The plug-in is identified by location. The mainframe plug-in slot nearest the display is slot A. Slot B is to the right of A.

Use the plug-in prefix *only* when a command operates on the plug-in controls, such as when setting the timebase. To set the timebase to 5 msec per division for the plug-in in slot A, for example, use the command:

A:TIME_DIV 5ms

Channel

The input channel is identified by its location in the plug-in. The uppermost left BNC connector in plug-in A is labeled A1. The next lower connector is A2, and so on. To modify all channels, do not specify a specific channel.

Use the channel prefix *only* for commands which affect vertical amplifiers, such as when setting the vertical sensitivity. To set the vertical sensitivity to 5 mV per division for plug-in A, channel 1, for example, use the command:

A1:VOLT_DIV 5mV

To set all of the plug-in's channel settings to be the same, use the plug-in prefix with no channel designation. For example, to set offset to 10 mV for all channels, use the command:

A:OFST 10 mV

Source

Since a trigger command can specify settings for each trigger source, it must be specified as a prefix. The prefix symbols are either:

An input channel as previously defined;
Plug-in and "EX" for the external trigger signal;
Plug-in and "EX10" for the external trigger signal divided by ten; or
Plug-in and "LINE" for triggering on the power line frequency.

To set the trigger level to 5 mV per division for the external trigger signal divided by ten, for example, use the command:

AEX10:TRIG_LEVEL 5mV

Trace

Traces 1 through 8, indicated as T1 through T8, define the processing and display characteristics of traces. For example, to query the horizontal position of trace 7 use the command

T7:HOR_POSITION?

System Header

This type of header indicates an action that affects general oscilloscope operation; that is, an operation not necessarily restricted to a particular plug-in, channel, or trace. Examples are changing the grid selection and cursor type. The System Header format disallows a prefix. For example, to turn on local display processing, use the command:

```
DISPLAY_ON
```

Standard Header

This type of header indicates a command that is explicitly required by the IEEE-488.2 standard. These commands have the same format as the System Header, except an asterisk ("*") immediately precedes the first letter of the command. For example, the *RST command initiates a device reset, *IDN? asks the device for its identification.

Command Argument(s)

A command argument(s) qualifies or supplements the header. It is included in the command only if a command is defined to require an argument(s). For example, an argument indicates what value to set for volts per division.

Most commands require one or more arguments to describe a desired action in detail (see Figure 2.4). The first argument is separated from the header by one or more spaces. Arguments are separated from each other by commas.

The possible types or arguments are:

Decimal Numeric

Any number in numeric repesentations NR1, NR2, or NR3 as defined by ANSI X3.42-1975. These refer to integers (e.g., -45), floating point (e.g., 3.1443), or exponential values (e.g., 3.1459E+00), respectively.

The ASCII characters "E" or "e" are used to delimit the mantissa from the exponent in exponential arguments. Spaces are allowed between the exponential delimiter and the digits (0 through 9), but are not allowed between digits, or between the decimal point (.) and the digits.

Numeric values with fractional parts must be expressed as a floating point or an exponental value. For example, 3.14159 and 3.14159E+00 are both acceptable standard formats.

The allowable range depends on the command. If a numeric is sent to the 7200A and has a precision greater than allowed, the 7200A will truncate, process the result, and generate a warning. If a numeric not included in the specified set is sent, a valid numeric closest to that sent is used. For example, vertical position must be specified with a value that is a multiple of 0.02. If 68.01 were sent, 68.00 is used.

Suffixes can replace exponential notation. For example, these commands are equivalent:

TIME_DIV 5.00E-6
TIME_DIV 5.00 US

Valid suffixes are listed in the following table:

Allowed<Suffix Mult.>Mnemonics

Definition	Mnemonics
1E18	EX
1E15	PE
1E12	T
1E9	G
1E6	MA
1E3	K
1E-3	M
1E-6	U
1E-9	N
1E-12	P
1E-15	F
1E-18	A

NOTE: Only engineering unit multipliers are allowed.

Table taken from ANSI/IEEE Std 488.2-1987

Table 2.1: Valid Suffix Mnemonics

Non-Decimal Numeric

Numbers can also be used in bases other than base 10. For example, each bit in a status byte may be understood better if the byte is specified in hexadecimal or binary.

Precede non-decimal arguments with a pound sign ("#") followed by an "H" for hexadecimal, "Q" for octal, and "B" for binary. Note that numbers are treated as unsigned with an implicit radix point. For example, #HFF and #B11111111 are both 255.

String

Some commands require or allow String arguments such as "ON" or "OFF". These arguments contain 7-bit ASCII alphanumeric characters. A string begins with an alpha character which may be followed by alphanumeric characters A through Z, a through z, 0 through 9, and "_". Carriage return or line feeds are not valid characters. Note that the oscilloscope treats upper and lower case characters identically.

Each command definition specifies the maximum number of characters for its string argument(s).

Quoted strings are delimited by either an apostrophe (') or a quotation mark ("). The 7200A returns quoted strings with quotation marks. The same type of delimiter that opens a quoted string must close it. Strings within strings are allowed as long as each string has the same opening and closing delimiters.

A quoted string may not be terminated with an **<end>** character. For example, **"test <end>** is an invalid string.

Waveform Data

The WAVEFORM command lets you send or read a complete waveform. The waveform usually contains a very large amount of data. Due to its length, a special formatting convention is used to transfer the large data blocks. See page 2-17 "Waveform Data Syntax" for an explanation of this format.

Keywords

Some commands have several arguments. For example, the command for configuring a hardcopy device can have up to ten arguments that set such characteristics as plotter speed and paper size. Rather than listing every argument when only a few need changing, the argument specified is identified by a "keyword".

A keyword is a character argument that must be followed by a comma, and then an associated value for the characteristic being set. The value may be one of the four types of arguments previously described.

Arguments not specified remain unaffected.

For example, the INTENSITY command is used to program the brightness of the traces and grids independently. To set the grid's intensity to half scale, send: **INTENSITY GRID,50**. The trace intensity remains unchanged.

For commands with several keywords, the order of the keyword-value pairs does not matter.

> *NOTE: A multi-argument command that does* ***not*** *use keywords must have all its arguments listed, and ordered in the same sequence as shown in the command definition.*

An example of a command with no keywords is the date command, which requires specification of the day followed by the month, etc.

When querying most keyword commands, you can give the keyword(s) as an argument. For example, **CRST? HABS, VABS** returns only the absolute horizontal and vertical cursor positions. If no argument is specified, all values are returned.

Command Terminators

Commands sent one at a time must end with a terminator. In this manual, terminators are indicated by **<end>**. Alternatively, multiple commands can be sent together by terminating each command with a semicolon and terminating the entire multiple command message with **<end>**.

GPIB Terminators

The only valid GPIB terminator is EOI (End or Identify) asserted with the last character sent. This is necessary because of the possibility of binary data transmission into the 7200A would make termination on a line feed alone impossible.

> *NOTE: The 7200A always terminates its response messages with a line feed character with EOI asserted.*

RS-232-C Terminators

The COMM_RS232 command is used to define the **<end>** for command messages, and separately for response messages.

The keyword EI defines the **<end>** of command messages as a number from 1 through 127. The default value is carriage return (decimal 13). If another value is selected, it must not be used elsewhere in the command argument; otherwise, the 7200A will prematurely terminate the command.

The keyword EO defines the end of messages transmitted by the 7200A. The initial value is CR LF (carriage return, then line feed).

For example, Arbitrary Block Program Data suitable for waveform transfers is sent as ASCII alphanumeric characters between the range 0 through 9 and A through F. Therefore, select a number that does not have a value corresponding to the ASCII decimal values of these alphanumeric characters.

Response Syntax

Query commands sent to the 7200A result in information being returned. The packet of returned information is called a response message. This message typically contains measurement results, settings, or status information. If multiple queries are sent in the same command, the responses will be returned in one multi-response message with the individual responses separated by semicolons (";").

The computer should completely read any responses from the 7200A before sending new queries. If the computer sends a query command, starts reading the response, and issues another command before completely reading the results of the first query, the 7200A interprets this as an interrupted action, sets the query error status bit, clears the output queue,and sends the second response.

Responses conform to a general format, and with few exceptions are ASCII strings of printable characters. Generally, the syntax of any response is as follows:

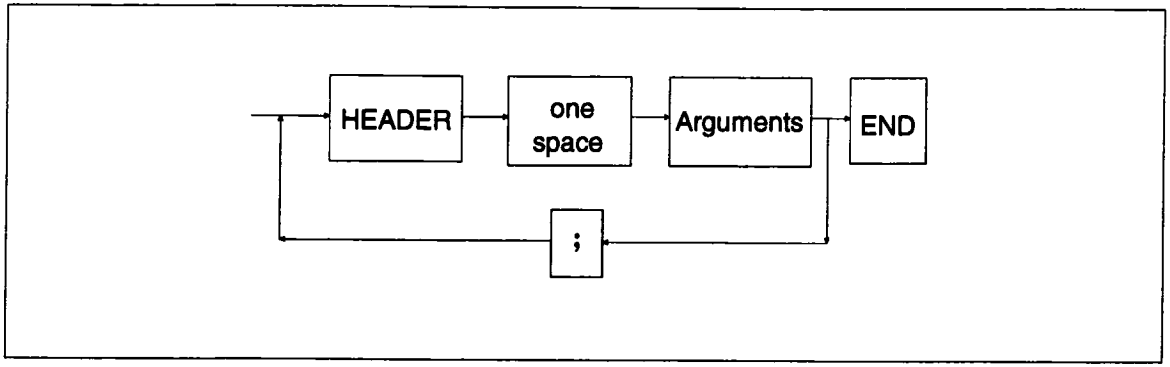


Figure 2.7: Response Syntax

where

<header> The syntax of a response header is the same as for a command header (see page 2-9). Prefixes are supplied when applicable. The header can be returned in either short, long, or no header (OFF) format as specified by the command COMM_HEADER (see page 5-21). Short format produces an adequate response for most circumstances. The long format yields the header in full English format for increased legibility. The OFF format achieves the fastest response time and requires little or no parsing.

<argument data> This part contains the information requested by the query command. It is separated from the header by one space. The argument data can be any of the types described in the Command Arguments section, p. 2-11.

<end> For GPIB, the **<end>** of a single or a multi-response message is always a line feed sent with EOI asserted.

For RS-232-C, **<end>** is the current setting of the EO argument of the COMM_RS232 command which is defined as a string. The default string is a CR (carriage return) followed by a LF (line feed). These are decimal 13 followed by decimal 10.

If the <end> string is contained within the response message, the computer may prematurely terminate the response from the 7200A. Programming <end> is provided for flexible response formats but must be used with caution.

If COMM_HEADER SHORT was sent before the following multi-response message:

T1:HOR_POSITION?; *STB?; T1:VERT_POSITION?<end>

The response would be:

T1:HPOS 1.0,1; *STB 48; VPOS 1.0<end>

Waveform Data Syntax

Waveform data is a specially formatted argument used to transfer large amounts of binary or hexadecimal data. The WAVEFORM command uses this argument to send data to, or to read data from the 7200A.

The 7200A supports three block data formats:

- Definite length arbitrary block data,
- Indefinite length arbitrary block data, and
- "OFF".

The COMM_FORMAT command (see page 5-19) is used to select the desired format.

Waveform Data transfers are used by the 7200A to send waveforms to a host controller. The waveform query will be used in this section to describe the waveform data response format. The WF? waveform query may optionally be followed by ALL, DESC, TEXT, DAT1, and DAT2 to query specific parts of the waveform. If nothing follows the WF?, then ALL is assumed.

Data Element Format

Each waveform point is called a data element. Using two commands, COMM_FORMAT and COMM_ORDER, the 7200A supports several methods of forming data elements. You can specify:

- the size or the width of the data element (i.e., the number of bytes),
- how it is encoded (i.e., binary or hexadecimal) and
- the arrangement of bytes for multi-byte words.

Data Width

The COMM_FORMAT command is used to select the size or width of the data element; that is, the number of 8-bit data bytes used to format a data element.

NOTE: Internal to the 7200A, waveforms are kept as 16-bit signed numbers. During readout to the computer, the 7200A converts the internal data words to one of:

BYTE 8 bits, single byte per data element.

WORD 16 bits, 2 bytes per data element.

Data Encoding

The block data bytes are either 8-bit binary or hexadecimal encoding. Use the COMM_FORMAT command to select the encoding.

BINary Binary format, the most compact, provides the fastest transfer.

NOTE: Binary format cannot be used over RS-232-C.

HEXadecimal Hexadecimal format . . . . .These numbers are indicated as ASCII characters between "0" through "9" and "A" through "F". Two characters represent each byte of data. The four most significant bits are represented by the first hexadecimal character, and the four least significant bits by the second. Commas do not separate the characters within a byte or between bytes. Therefore, to separate the values into bytes, the computer uses the data encoding specified by the COMM_FORMAT command.

A block of eight bytes, for example, is received in this format as follows:

HEXADECIMAL	ASCII
45 41 30 46 31 33 30 34 0d 0a	**EA0F1304<end>**

where EA, OF, 13, 04 are the hexadecimal values for four successive 8-bit data bytes. (In decimal notation these values correspond to 234, 15, 19, and 04 respectively.) In this example, **<end>** is CR followed by LF (or 13 followed by 10 in decimal).

Byte Order

When the width of the data is words, the order or sequence in which the bytes are sent can be selected using the COMM_ORDER command:

HI	The most significant byte is sent first, the least significant is sent last. This format is generally known as the Motorola format since the most significant byte is at a lower address.
LO	The least significant byte is sent first, the most significant is sent last. The LO format is known as the Intel (or Zilog) format since the least significant byte is at a lower address.

Definite length arbitrary block data

This format is used for sending blocks of previously fixed sizes. It is selected with the **COMM_FORMAT DEF9** comand:

#9nnnnnnnnn<DB1><DB2>...<DBX>

where **nnnnnnnnn** is a decimal integer defined by nine bytes and is used to define the number of data bytes.

<DB1>,<DB2>,...,<DBX> are 8-bit data bytes.

In this format, a block of four bytes, for example, is sent or received

as follows:

#9000000004<DB1><DB2<DB3><DB4>

This format begins with a pound sign ("#") followed by a "9", then 9 bytes (which when taken together form a decimal integer equal to the number of 8-bit data bytes to follow), and the four 8-bit data bytes.

If the waveform data is the last command of the message response, the command terminator follows the waveform data. (The integer following "#9" does *not* count this terminator); otherwise, it is separated from the next command or response by a semicolon.

Indefinite length arbitrary block data

This format is used to send blocks of unspecified length. It is selected with the **COMM_FORMAT IND0** command:

#0<DB1><DB2>...<DBX><end>

where **<end>** is a previously defined message terminator.

The format begins with a pound sign ("#") followed by a "0", 8-bit data bytes, and the message terminator.

Note that since the number of bytes is not known and the data is binary,it would not be possible to separate it from another command or response and therefore *MUST* be followed by <end>.

If the WF? is sent as a compound string with queries following it (i.e., T1:WF?;T2:TRACE?), the query error bit will be set and any queries following the WF? will be ignored. The response message will contain any responses to queries preceding WF? followed by the response to WF?

OFF Format

Some computer languages make it difficult to check a few bytes/characters before receiving the remainder of a data block. Therefore, the 7200A has the "OFF" format which is an extension of IEEE Standard 488.2.

The OFF format is nearly identical to #0 indefinite format, except that it supresses the #0, keywords normally included in the response, and comma separators. It is selected with the **COMM_FORMAT OFF** command.

When OFF format is specified, the 7200A returns a data block having the format:

<DB1><DB2>...<DBX><END>

As with the #0 format above, the OFF block must be terminated with <end>.

> *NOTE: When writing waveforms back to the scope, the only accepted formats are #0 and #9, and the descriptor must be sent with the data.*

Example: Given that Trace 1 consists of a 338 byte descriptor and 1000 bytes for data array 1, if the block format is OFF, COMM_HEADER is OFF, and the 7200A is sent the query:

T1:WF?<end>

the 7200A response would consist of 338 bytes for the descriptor plus 1000 bytes for the wave array:

<338 byte DESCRIPTOR><DB339><DB340>...<DB1337><DB1338><end>

which is all data. Note, however, if the header was short or long, the prefix followed by the alias or command name, respectively, would preceed the data.

To write this data back to the 7200A, precede it with the command header (T1:WAVEFORM), its keyword (ALL), comma, and #0 prior to transmitting the data:

T1:WAVEFORM ALL,#0<338 byte DESCRIPTOR> <DB339> <DB340>...<DB1337><DB1338><end>

Alternatively, the #9 format may be used:

T1:WAVEFORM ALL,#9000001338<338 byte DESCRIPTOR> <DB339> <DB340> ....<DB1337><DB1338><end>

RS-232-C Output Format

Waveform data over GPIB can be encoded in either binary or hexadecimal (see COMM_FORMAT command). However, over RS-232-C the data encoding must be set to HEX.

To format the appearance of the waveform data while printing on an RS-232-C device, use the COMM_RS232 command to specify the maximum number of characters per line. The keyword, LL, followed by a # specifies the maximum line length. After the end of each line, the designated line separator character is inserted. This character is defined by the keyword LS in the COMM_RS232 command.

For the count in the Definite length arbitrary block data, the line separator is not counted.

Also, for block data sent to the 7200A, LS is ignored.

For example, the response to the query **T1:WAVEFORM? DAT1** can be a block of 32 bytes of data received from the 7200A's RS-232-C host port. It is received in definite length arbitrary block format in hexadecimal encoding (required for RS-232-C):

T1:WAVEFORM DAT1,#900000003212345678901234567890123456<Line_Separator> DC78EF87DF0C128A<Line_Separator><end>

The "#9000000032" indicates the format ("#9") and the number of bytes sent (32 bytes excluding the LS and the <end>). Note that HEX format doubles the number of binary bytes representing the waveform. In this example, 32 HEX bytes bytes are transferred but really represent 16 8-bit bytes of trace 1 (i.e., "1", "2" represents 12 base 16 or 18 decimal). Note also that the waveform preamble (T1:WAVEFORM DAT1,#9000000032) is ALWAYS ASCII. This is always true if the data encoding is HEX or BINary. Also note that the line length (LL) is 44 characters per line. When the <end> is reached before the LL is reached, an LS is sent before <end> is sent.

Section 3: Waveform Transfer

Waveforms can be transferred between the 7200A and an external device via GPIB, RS-232-C, or MSDOS format 3-1/2" floppy disk. All types of transmission use the same format for the waveform. It is therefore possible, for example, to read a waveform out of the 7200A over GPIB, direct the output into a file on an MSDOS floppy, and then recall the waveform from the floppy to a memory in the 7200A.

Over GPIB or RS-232-C, the WAVEFORM remote command transfers a binary waveform from an external device into the 7200A. The WAVEFORM? query transfers a binary waveform from the 7200A to an external device. The COMM_FORMAT and COMM_ORDER remote commands select the data point format to be used by the 7200A when it produces waveforms in response to a WAVEFORM? query. The INSPECT? query transfers an ASCII waveform from the 7200A to an external device.

Waveform transfer via floppy disk does not require remote programming, but may be accomplished with STORE and RECALL front panel operations. Remote commands STORE_SETUP, STORE, RECALL_SETUP, and RECALL may also be used to effect the store and recall operations if desired.

Waveform Template

Waveforms produced by the 7200A contain, in addition to the actual data points, further information necessary to correctly interpret the data. This information includes the real time between data points, trigger offset, vertical gain and offset, acquisition time and plugin, etc. To save space and increase the waveform transfer rate, all numerical values in the waveform are binary.

The data and associated information are organized in a specific format described by the waveform template. The template describes the size and location of each element in the waveform. It may be obtained via GPIB or RS-232-C with the TEMPLATE? query. The template is simply an ASCII file and may be examined with any text editor. See page 3-9 for a listing of the 7200A waveform template. On each line of the template, text following a ; is commentary.

In addition to providing a description of the waveform structure to users who wish to interpret waveforms obtained from the 7200A, the template also allows waveforms to be transferred between different LeCroy instruments and different versions of the same instrument. Waveform transferral between different LeCroy instruments may be accomplished by sending not only the waveform, but also the template according to which it was created, to the destination instrument. The destination instrument interprets the waveform data according to the associated template and translates it into its own format.

As an alternative to using the template to interpret a waveform, the INSPECT? query may be used to obtain a nicely formatted and labeled ASCII representation of the waveform.

As can be seen from the template on page 3-9, a 7200A waveform consists of several distinct entities called blocks:

Waveform Descriptor — Contains information necessary to correctly interpret the waveform data. The waveform descriptor contains two types of information:

1. Identification and description of the waveform format (name of the associated template, length of each block present in the waveform, etc.)
2. Information associated with the waveform data points (time per point, vertical gain, etc.)

User Text — This optional block may contain user documentation.

Sequence Trigger Times — This block is present only for sequence waveforms and contains the trigger time and trigger offset for each segment of the waveform.

Wave Array 1 — Contains the actual data points comprising the waveform

Wave Array 2 — This block is present only for dual waveforms (produced by the EXTREMA or FFTRI functions) and contains the second data array.

Each block contains distinct elements called fields. The template gives a detailed description of the fields comprising each block. Each field is described by a line of the form:

< offset> name:type ;comment

where

offset — is the decimal offset in bytes of the field relative to the beginning of the block. (The first field in a block has an offset of zero.)

name — is the name of the field

type — identifies the data format used to represent the field's value. All numerical values in the waveform are binary and must be interpreted according to their specified type. The possible types are described in the comments at the beginning of the waveform template on page 3-9.

For example, refer to page 3-11 of the waveform template. The value of the vertical gain is contained in the field named VERTICAL_GAIN, which is located 120 bytes from the start of the waveform descriptor block and is represented as a 32-bit IEEE format single precision floating point number.

Interpreting A Waveform

On page 3-4 is a hexadecimal/ASCII dump of an example waveform produced by the 7200A in response to a WAVEFORM? query. The waveform descriptor block is located at the beginning of the waveform and starts with the character string WAVEDESC. This is followed by the character string name of the template which describes the waveform's organization. For the example waveform, the template name is LECROY_1_0".

To find the value of a field in the waveform descriptor, first find its offset and type in the template. The offset specifies where to find the value in the waveform itself, and the type specifies the size of the value and how to interpret it. Then retrieve the value from the waveform and interpret it according to its type.

The byte order must also be known in order to correctly interpret numerical values. Two alternate byte orders are possible:

most significant byte .. least significant byte (Motorola format) or,

least significant byte .. most significant byte (Intel format).

The byte order of the waveform is specified by the COMM_ORDER field in the waveform descriptor. According to page 3-10 of the template, COMM_ORDER is a 16-bit value (type enum) located at offset 34 from the start of the waveform descriptor block. Because the waveform descriptor is the first block in the waveform, COMM_ORDER is the 16-bit value located at offset 34 from the start of the waveform. This value is $0000_{(16)}$ or $0_{(10)}$. The description of the COMM_ORDER field in the template indicates that value 0 is the code meaning HIFIRST. This specifies the byte ordering of all numerical values in the waveform to be most significant byte .. least significant byte.

Once the byte order is known, any field in the waveform descriptor can be interpreted from its offset and type specified in the template. For example, the field NOMINAL_BITS is represented as a 16-bit value (type word) located at offset 132 from the start of the waveform descriptor. The 16-bit value located at offset 132 with byte ordering most significant ... least significant is $0008_{(16)}$. Interpreting these 16 bits as a 2's complement signed number (as specified by type word) yields 8.

OFFSET	HEXADECIMAL															ASCII
000000	57	41	56	45	44	45	53	43	00	00	00	00	00	00	00	00	WAVEDESC........
000016	4c	45	43	52	4f	59	5f	31	5f	30	00	00	00	00	00	00	LECROY_1_0......
000032	00	01	00	00	00	00	01	38	00	00	00	00	00	00	00	20	.......8........
000048	00	00	00	36	00	00	00	00	4c	65	43	72	6f	79	20	37	...6....LeCroy 7
000064	00	00	00	00	00	00	00	00	00	00	1c	20	54	72	61	63	200 DSO..... Trac
000080	65	31	00	00	00	00	00	00	00	00	00	00	00	00	00	68	e1.............h
000096	00	00	00	64	00	00	00	00	00	00	00	67	00	00	00	02	...d.......g....
000112	00	00	00	02	00	00	00	01	38	80	00	00	00	00	00	00	........8.......
000128	7f	00	80	00	00	08	30	89	70	5f	be	5b	79	08	ae	00	......0.p_.[y...
000144	00	00	be	5a	d7	f2	8e	00	00	00	56	00	00	00	00	00	...Z......V.....
000160	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	....
000176	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	....
000192	00	00	00	00	00	00	00	00	00	00	53	00	00	00	00	00	..........S.....
000208	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	....
000224	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	....
000240	00	00	00	00	00	00	00	00	00	00	40	08	3d	70	a3	d7	..........@.=p..
000256	0a	3d	33	09	01	01	07	c5	1e	ea	00	00	00	00	00	00	.=3.............
000272	00	00	00	0b	00	00	3f	80	00	00	00	11	00	00	3f	80	......?.......?.
000288	00	00	00	00	00	00	00	02	00	01	00	00	00	03	00	00	....
000304	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	....
000320	be	5b	79	08	ae	00	00	00	3f	0e	77	a9	0f	40	00	00	.[y.....?.w..@..
000336	be	5b	c1	a7	72	00	00	00	ba	c2	bb	d4	bb	cc	bb	d0	.[..r...........
000352	bb	c2	bb	ce	bb	ca	bb	ce	bb	cc	bb	cc	ba	ba	bb	cc	....
000368	bb	ca	bc	ba	bb	bc	bb	ca	bc	d0	bc	d0	bb	c8	c0	0a	....
000384	c3	30	c9	c2	cf	0e	d9	b8	e3	04	ef	0e	f8	3c	02	16	.0..............
000400	0b	32	15	a2	19	70	20	4a	27	2e	29	e4	2b	5c	2f	06	.2...p J'.).+\/..
000416	2f	e6	2d	be	2d	fa	30	76	2e	50	2f	26	2e	1a	2f	56	/.-.-.0v.P/&.../V
000432	2e	34	2f	3c	2e	1e	2f	4c	30	4a	32	36	32	72	34	9a	.4/../L0J262r4..
000448	bc	f4	bb	be	bc	f6	bb	bc	bc	e8	bb	cc	bc	d6	bb	bc	....
000464	bc	e8	bb	cc	bc	d6	bb	bc	bc	e8	bb	cc	bc	d6	bc	ce	....
000480	bd	fa	bc	de	be	e8	bf	00	c4	68	c9	86	d1	f8	d9	da	.........h......
000496	e4	84	ed	cc	f8	ea	02	2a	0b	06	13	0c	1d	94	20	26	.......*....... &
000512	28	04	2a	b4	2c	94	2d	92	31	6e	30	22	2f	02	2f	14	.*.,.-..1n0"/./.
000528	2f	74	2f	5c	2f	46	2e	32	2e	36	2f	44	30	3c	30	58	/t/\/F.2.6/D0X
000544	32	6e	31	64	34	62	33	90									2n1d4b3.

Figure 3.1: Hexadecimal/ASCII dump of a waveform

Retrieving the data points of a waveform requires several steps:

1. **Locate the start of the data in the waveform**

   To locate the beginning of wave array 1 in the waveform, add up the lengths of the blocks which precede it " waveform descriptor, user text, and sequence trigger times.

   Page 3-10 of the template specifies that the length in bytes of the waveform descriptor block is contained in the field named WAVE_DESCRIPTOR, which is a 32-bit value (type long) located at offset 36 from the start of the waveform descriptor block. For the example, the 32-bit value located at offset 36 from the start of the waveform descriptor block (which is the first block in the waveform) is $(00000138)_{16}$, or $(312)_{10}$. Since the value of the WAVE_DESCRIPTOR field is $(312)_{10}$, the waveform descriptor block is $(312)_{10}$ bytes long. Page 3-10 of the template specifies that the length in bytes of the user text block is contained in the field named USER_TEXT, which is a 32-bit value located at offset 40 from the start of the waveform descriptor block. For the example waveform, the field USER_TEXT in the waveform descriptor is 0. This means that the user text is not present in this waveform. Similarly, for the example waveform the sequence trigger time array (TRIGTIME_ARRAY) is 32 bytes long.

   For the example waveform, the data array (wave array 1) starts at offset 312 + 0 + 32 = 344 from the beginning of the waveform.

2. **Determine the number of data points present.**

   The total number of points in the data array is specified by the WAVE_ARRAY_COUNT field in the waveform descriptor. Page 3-11 of the template indicates that this field is a 32-bit value located at offset $(92)_{10}$. For the example waveform, WAVE_ARRAY_COUNT has the value of $(00000068)_{16}$ or $(104)_{10}$.

   Not every point in the data array may be valid, however. For example, if the waveform corresponds to a horizontally repositioned trace it may be missing some points off one end. The FIRST_VALID_PNT and LAST_VALID_PNT fields in the waveform descriptor give the indices (starting from 0) of the first and last valid data points in the data array. FIRST_VALID_PNT and LAST_VALID_PNT are 32-bit values located at offsets $(100)_{10}$ and $(104)_{10}$ respectively. For the example waveform, FIRST_VALID_PNT is 0 and LAST_VALID_PNT is $(00000067)_{16}$ or $(103)_{10}$. Because WAVE_ARRAY_COUNT = (Last_Valid_Point - First_Valid_Point + 1), every point in the data array must be valid for this particular waveform.

3. **Determine the format or representation of each data point.**

   According to page 3-16 of the template, WAVE_ARRAY_1 consists of an array of measurement values or data points whose data format is described by the COMM_TYPE field in the waveform descriptor block.

The COMM_TYPE field specifies the size and type of data representation of each data point.

Page 3-10 of the template specifies that the COMM_TYPE field is a 16-bit value (type enum), located at offset 32 within the waveform descriptor . For the example waveform, the COMM_TYPE field has the value 1. The description of the COMM_TYPE field in the template indicates that value 1 is the code meaning word. This specifies that each point in the data array is of type word, which is a 16-bit 2's complement signed data representation.

The first three (unscaled) points of the data array of the example waveform are $(BAC2)_{16}$, $(BBD4)_{16}$, and $(BBCC)_{16}$ or $(-17726)_{10}$, $(-17452)_{10}$, and $(-17460)_{10}$.

4. **Scale each data point value.**

Further information in the waveform descriptor is used to correctly scale the vertical magnitude of each point in the data array.

The units of the vertical value are given by the VERTUNIT field of the waveform descriptor, which is an ASCII character string located at offset 154. For the example waveform, the vertical units are V, or volts.

The vertical value V[i] of each data point i is given by the following equation:

V[i] = data[i] * VERTICAL_GAIN - VERTICAL_OFFSET

where data[i] is the ith point in the data array, and VERTICAL_GAIN and VERTICAL_OFFSET are fields in the waveform descriptor .

Page 3-11 of the template indicates that VERTICAL_GAIN and VERTICAL_OFFSET are 32-bit IEEE format floating point values located at offsets 120 and 124, respectively. For the example waveform, the VERTICAL_GAIN is $(38800000)_{16}$, which corresponds to the value $(6.1E\text{-}5)_{10}$. The VERTICAL_OFFSET is 0.

5. **Determine the horizontal coordinate of each data point.**

The units of the horizontal coordinate are given by the HORUNIT field of the waveform descriptor, which is an ASCII character string located at offset 202. For the example waveform, the horizontal units are s, or seconds.

The horizontal coordinate of each data point depends on whether or not the waveform is a sequence waveform. This can be determined from the NOM-SUBARRAY_CNT field in the waveform descriptor, which specifies the nominal number of segments in the waveform. If NOM_SUBARRAY_CNT is greater than 1, the waveform is a sequence waveform. NOM_SUB ARRAY_CNT is a 32-bit value (type long) located at offset 112. For the sample waveform, NOM_SUBARRAY_CNT has the value 2. Thus the waveform is a sequence waveform.

5.1. Non-Sequence Waveforms

The horizontal coordinate T[i] of data point i is given by the following equation:

T[i] = i * HORIZ_INTERVAL + HORIZ_OFFSET

where i = 0..(WAVE_ARRAY_COUNT-1)

HORIZ_INTERVAL and HORIZ_OFFSET are fields in the waveform descriptor .

For time-domain waveforms, HORIZ_OFFSET is the time in seconds from the trigger to the first data point (it may be negative) and HORIZ_INTERVAL is the time between data points, or the sampling interval. The horizontal coordinate T[i] is thus the time relative to trigger of data point i.

For frequency-domain or histogram waveforms, the above equation will yield the horizontal coordinate T[i] as absolute frequency or bin location, respectively, with units given by the HORUNIT field.

5.2. Sequence Waveforms

Sequence waveforms are composed of several consecutive segments, each of which has its own trigger time and trigger offset. The wave array block of sequence waveforms contains all the segments. The actual number of segments present is given by the SUBARRAY_COUNT field in the waveform descriptor. Note that the actual number of segments present may be less than or equal to the nominal number given by NOM_SUBARRAY_CNT. The number of data points in each segment can be calculated as follows:

Data points per segment = WAVE_ARRAY_COUNT/NOM_SUBARRAY_CNT.

Sequence waveforms contain a sequence trigger time block which is an array of trigger time and trigger offset for each segment. Page 3-16 of the template describes the sequence trigger time block. The block structure contains two fields, the TRIGGER_TIME and the TRIGGER_OFFSET. In the waveform, these two fields are repeated for each segment to comprise the sequence trigger time block.

For each segment, the corresponding TRIGGER_TIME field in the sequence trigger time block contains the time in seconds between the trigger of the first segment and the trigger of the current segment. (The time of the first trigger is given by the TRIGGER_TIME field in the waveform descriptor). The TRIGGER_OFFSET field contains the time in seconds from the trigger of the current segment to the first data point of the segment.

The example waveform is a sequence waveform with two segments (SUBARRAY_COUNT equals 2). The sequence trigger time block is located after the waveform descriptor and user text blocks, and therefore starts at offset 312+ 0= 312 from the beginning of the waveform (where 312 is the length in bytes of the waveform descriptor block and 0 is the length in bytes of the user text block, as determined above).

The trigger time block contains two consecutive repetitions of the {TRIGGER_TIME,TRIGGER_OFFSET} structure. According to the template, TRIGGER_TIME and TRIGGER_OFFSET are each 64-bit IEEE format double precision floating point values. The TRIGGER_OFFSET of the second segment is $(BE5BC1A77200A0000)_{16}$, or $(-2.6E-8)_{10}$.

The HORIZ_INTERVAL field in the waveform descriptor gives the time between data points, or the sampling interval, which is the same for each segment.

The horizontal time coordinate Trel[i,seg] of data point i in segment seg relative to the trigger for that segment is given by the following equation:

Trel[i,seg] = i * HORIZ_INTERVAL + TRIGGER_OFFSET[seg]

The horizontal time coordinate Tabs[i,seg] of data point i in segment seg relative to the first trigger is given by the following equation:

Tabs[i,seg] = Trel[i,seg] + TRIGGER_TIME[seg]

where seg = 0..(SUBARRAY_COUNT-1)
i = 0..(Data points per segment - 1)

Example Waveform Template

```
/00
000000              LECROY_1_0:  TEMPLATE
                5 54 130
;
; Description of the structure of waveforms produced by
; LeCroy Digital Oscilloscopes.
;
; A waveform consists of several logical data blocks whose formats are
; explained below.
; A complete waveform consists of the following blocks:
; the descriptor block WAVEDESC
; the text descriptor block USERTEXT (optional)
; the time array block (for sequence waveforms only)
; data array block
; auxiliary or second data array block (for dual waveforms only)
;
; In the following explanation, every element of a block is described by a
; single line in the form
;
; < byte position>   < variable name> : < variable type>  ; < comment>
;
; where
;
; < byte position>  =  position in bytes (decimal offset) of the variable,
; relative to the beginning of the block.
;
; < variable name>  =  name of the variable.
;
; < variable type>  =  string          up to 16-character name
;                                      terminated with a null byte
;                 byte                 8-bit signed data value
;                 word                 16-bit signed data value
;                 long                 32-bit signed data value
;                 float                32-bit IEEE floating point value
;                 double               64-bit IEEE floating point value
;                 enum                 enumerated value in the range 0 to N
;                                      represented as a 16-bit data value.
;                                      The list of values follows immediately.
;           time_stamp                 consists of the following fields:
;                                      double   seconds      0.00 to 59.999999)
;                                      byte     minutes      (0 to 59)
;                                      byte     hours        (0 to 23)
```

```
;                                   byte      days      1 to 31)
;                                   byte      months    (1 to 12)
;                                   word      year      (0 to 16000)
;                                   word      unused
;                                   (There are 16 bytes in a time field.)
;                 data              byte or word, as specified by the COMM_TYPE
;                                   variable in the WAVEDESC block
;                 text              arbitrary length text string (maximum 400)
;            unit_definition        48 character null-terminated ASCII string
;                                   for the unit name.
;= = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = =
;
WAVEDESC: BLOCK
;
; Explanation of the wave descriptor block  WAVEDESC ;
;
< 0>  DESCRIPTOR_NAME: string  ; the first 8 chars are always WAVEDESC
;
< 16>      TEMPLATE_NAME: string
;
< 32>      COMM_TYPE: enum
           _0   byte
           _1   word
           endenum
;
< 34>      COMM_ORDER: enum
           _0   HIFIRST
           _1   LOFIRST
           endenum
;
;
; The following variables specify the block lengths of all blocks of which
; the entire waveform (as it is currently being read) is composed.
; If a block length is zero, the block is (currently) not present.
;
;
;BLOCKS :
;
< 36>      WAVE_DESCRIPTOR: long          ; length in bytes of block WAVEDESC
< 40>      USER_TEXT: long                ; length in bytes of block USERTEXT
;
;ARRAYS :
;
< 44>      TRIGTIME_ARRAY:long            ; length in bytes of TRIGTIME array
;
< 48>      WAVE_ARRAY_1: long             ; length in bytes of 1st data array
;
< 52>      WAVE_ARRAY_2: long             ; length in bytes of 2nd data array
;
;
; The following variables identify the instrument
```

```
;
< 56>      INSTRUMENT_NAME: string
;
< 72>      INSTRUMENT_NUMBER: long
;
; The following variables describe the waveform type and the time at
; which the waveform was generated.
;
< 76>      TRACE_LABEL: string              ; trace label
;
< 92>      WAVE_ARRAY_COUNT: long           ; actual number of points in each
                                            ; data array
;
< 96>      PNTS_PER_SCREEN: long            ; nominal number of points in each
                                            ; data array
;
< 100>     FIRST_VALID_PNT: long            ;count of number of points to skip
                                            ; before first good point in data
                                            ; array,FIRST_VALID_POINT =  0
                                            ; for normal waveforms.
;
< 104>     LAST_VALID_PNT: long             ; index of last good data point
                                            ; in data array
;
< 108>     SUBARRAY_COUNT: long             ; for sequence waveforms,actual number
                                            ; of segments in data array
;
< 112>     NOM_SUBARRAY_CNT: long           ; nominal number of segments in data
                                            ; array
                                            ; NOM_SUBARRAY_CNT:=  1
                                            ; for non-sequence waveforms
< 116>     SWEEPS_PER_ACQ: long             ; for cumulative waveforms (eg.average),
                                            ; number of sweeps which contributed to
                                            ; the waveform
;
< 120>     VERTICAL_GAIN: float
;
< 124>     VERTICAL_OFFSET: float           ; to get floating values from raw data
                                            ; VERTICAL_GAIN*data-VERTICAL_OFFSET
;
< 128>     MAX_VALUE: word                  ; maximum allowed data value
;
< 130>     MIN_VALUE: word                  ; minimum allowed data value
;
< 132>     NOMINAL_BITS: word               ; a measure of the intrinsic precision
                                            ; of the observation: ADC data is 8 bit
                                            ; averaged data is 10-12 bit, etc.
;
```

```
< 134>    HORIZ_INTERVAL: float          ; sampling interval for time domain
                                         ; waveforms
;
< 138>    HORIZ_OFFSET: double           ; actual trigger delay (time in seconds
                                         ; from trigger to first data point)
;
< 146>    PIXEL_OFFSET: double           ; nominal trigger delay selected by
                                         ; user
;
< 154>    VERTUNIT: unit_definition      ; units of the vertical axis
;
< 202>    HORUNIT: unit_definition       ; units of the horizontal axis
;
;
; The following variables describe the time at which
; the waveform was generated and the waveform type
;
< 250>    TRIGGER_TIME: time_stamp       ; time of the trigger
;
< 266>    ACQ_DURATION: float            ; duration of the acquisition (in sec)
                                         ; in multi-trigger waveforms.
                                         : (e.g. sequence or averaging)
;
< 270>    RECORD_TYPE: enum
              _0      single_sweep
              _1      interleaved
              _2      histogram
              _3      trend
              _4      filter_coefficient
              _5      complex_frequency_domain
              _6      extrema-_envelope_display
              _7      sequence
              endenum
;
< 272>    PROCESSING_DONE: enum
              _0      no_processing
              _1      fir_filter
              _2      interpolated
              _3      sparsed_waveform
              _4      autoscaled
              _5      no_result
              _6      roll_mode
              _7      cumulative
              endenum
;
;
; The following variables describe the basic acquisition
; conditions used when the waveform was acquired
```

```
;
< 274>      TIMEBASE: enum
                    _0        1_ps/div
                    _1        2_ps/div
                    _2   5_ps/div
                    _3   10_ps/div
                    _4   20_ps/div
                    _5   50_ps/div
                    _6   100_ps/div
                    _7   200_ps/div
                    _8   500_ps/div
                    _9   1_ns/div
                    _10  2_ns/div
                    _11  5_ns/div
                    _12  10_ns/div
                    _13  20_ns/div
                    _14  50_ns/div
                    _15  100_ns/div
                    _16  200_ns/div
                    _17  500_ns/div
                    _18  1_us/div
                    _19  2_us/div
                    _20  5_us/div
                    _21  10_us/div
                    _22  20_us/div
                    _23  50_us/div
                    _24  100_us/div
                    _25  200_us/div
                    _26  500_us/div
                    _27  1_ms/div
                    _28  2_ms/div
                    _29  5_ms/div
                    _30  10_ms/div
                    _31  20_ms/div
                    _32  50_ms/div
                    _33  100_ms/div
                    _34  200_ms/div
                    _35  500_ms/div
                    _36  1_s/div
                    _37  2_s/div
                    _38  5_s/div
                    _39  10_s/div
                    _40  20_s/div
                    _41  50_s/div
                    _42  100_s/div
                    _43  200_s/div
                    _44  500_s/div
                    _45  1_ks/div
                    _46  2_ks/div
                    _47  5_ks/div
                    endenum
```

```
< 276>      VERT_COUPLING: enum
                _0   DC_50_Ohms
                _1   ground
                _2   DC_1MOhm
                _3   ground
                _4   AC,_1MOhm
                endenum
;
< 278>      PROBE_ATT: float
;
< 282>      FIXED_VERT_GAIN: enum
                _0   1_uV/div
                _1   2_uV/div
                _2   5_uV/div
                _3   10_uV/div
                _4   20_uV/div
                _5   50_uV/div
                _6   100_uV/div
                _7   200_uV/div
                _8   500_uV/div
                _9   1_mV/div
                _10  2_mV/div
                _11  5_mV/div
                _12  10_mV/div
                _13  20_mV/div
                _14  50_mV/div
                _15  100_mV/div
                _16  200_mV/div
                _17  500_mV/div
                _18  1_V/div
                _19  2_V/div
                _20  5_V/div
                _21  10_V/div
                _22  20_V/div
                _23  50_V/div
                _24  100_V/div
                _25  200_V/div
                _26  500_V/div
                _27  1_kV/div
                endenum
;
< 284>      BANDWIDTH_LIMIT: enum
                _0          off
                _1          on,_80_MHz
                endenum
;
< 286>      VERTICAL_VERNIER: float
;
< 290>      ACQ_VERT_OFFSET: float
;
< 294>      WAVE_SRC_PLUGIN: enum
```

```
                _0          UNKNOWN
                _1          PLUGIN_A
                _2          PLUGIN_B
                _3          PLUGIN_C
                _4          PLUGIN_D
                _5          PLUGIN_E
                _6          PLUGIN_F
                endenum
;
< 296>      WAVE_SRC_CHANNEL: enum
                _0          UNKNOWN
                _1          CHANNEL_1
                _2          CHANNEL_2
                _3          CHANNEL_3
                _4          CHANNEL_4
                endenum
;
< 298>      TRIGGER_SOURCE: enum
                _0          CHANNEL_1
                _1          CHANNEL_2
                _2          CHANNEL_3
                _3          CHANNEL_4
                _4          LINE
                _5          EXT
                _6          EXT/10
                endenum
;
< 300>      TRIGGER_COUPLING: enum
                _0          AC
                _1          LF_REJ
                _2          HF_REJ
                _3          DC
                endenum
< 302>      TRIGGER_SLOPE: enum
                _0          POSITIVE
                _1          NEGATIVE
                endenum
;
< 304>      SMART_TRIGGER: enum
                _0          OFF
                _1          ON
                endenum
;
< 306>      TRIGGER_LEVEL: float
;
< 310>      SWEEPS_ARRAY1: long          ;for alt sync avg waveforms,
                                         ;number of sweeps which contributes to
                                         ;the number in data array 1
;
```

```
< 314>     SWEEPS_ARRAY2: long          ;for alt sync avg waveforms,
                                        ;number of sweeps which contributed to
                                        ;the waveform in data array 1
;
/00    ENDBLOCK
;



;= = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = =
;
USERTEXT: BLOCK
;
; Explanation of the descriptor block USERTEXT at most 400 bytes long.
;
< 0>  DESCRIPTOR_NAME: string; the first 8 chars are always USERTEXT
;
<  16>      TEXT: text                  ; this is simply a list of ASCII
                                        : characters
;
/00          ENDBLOCK
;
;
;= = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = =
;
TRIGTIME: ARRAY
;
; Explanation of the trigger time array (present for sequence waveforms only)
;
< 0>  TRIGGER_TIME: double            ; time in seconds from first trigger
                                      ; to this one
;
< 8>  TRIGGER_OFFSET: double          ; actual trigger delay time in
                                      : seconds from this trigger to
                                      : first data point
;
/00          ENDARRAY
;
;
;= = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = =
;
WAVE_ARRAY_1: ARRAY
;
;
< 0>  MEASUREMENT: data               ; the actual format of a data is
                                      : given in the WAVEDESC descriptor
                                      ; by variable COMM_TYPE
; /00         ENDARRAY
;
;
;= = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = =
```

```
;
WAVE_ARRAY_2: ARRAY
;
;
< 0>  MEASUREMENT: data          ; the actual format of a data is
                                 ; given in the WAVEDESC descriptor
                                 ; by variable COMM_TYPE
;
/00          ENDARRAY
```

Section 4: Status Messages

This section describes the 7200A's instrument status and event reporting functions. Although the mechanisms for requesting service differ for GPIB and RS-232-C, status and event reporting is identical.

GPIB Service Request

When the 7200A reports a change in its condition, it can asynchronously request service from the GPIB controller (e.g., the 7200A can request service when processing completes). The 7200A requests service by asserting the GPIB Service Request (SRQ) management line. Once it has been enabled by setting the appropriate mask bits, the SRQ interrupts the controller.

To identify the source of the SRQ, the controller serial polls the devices attached to the GPIB. It reads the main Status Byte register (STB) of each device polled. To read the STB, the controller addresses a device to talk and sends it a Serial Poll Enable bus command. In return, the device sends its STB. The device whose STB has an asserted RQS bit (seventh bit) generated the SRQ.

Once the controller determines that the 7200A generated the SRQ, it will reset the SRQ line. The 7200A will then reset its RQS bit.

RS-232-C Service Request

The RS-232-C interface does not have a line by which the 7200A can asynchronously request service from the computer. Therefore, it must query (poll) the 7200A to read the STB register.

The computer issues the status command, *STB? to query the 7200A for its STB register. The seventh bit of the STB register (RQS bit for GPIB serial polling, MSS or Master Summary Status bit for STB) indicates that the 7200A requests service.

Status Byte Operation

The 7200A continually updates its status to report the latest events, conditions, and settings. Changes are summarized by designated bits in the Status Byte register (STB). The seventh bit, RQS, is asserted whenever any other bits in the STB are reported as set and their corresponding mask bits are enabled. Also, whenever the RQS bit is set, the GPIB bus SRQ line is automatically asserted.

Typically, the controller identifies the device requesting service. It reads the other bits in the STB register and the associated status data structure to determine the cause of, and the conditions relating to the service request. For example, whenever the sixth bit (ESB) of the STB

register is set, an error or synchronization event has occurred. The controller then examines the Event Status Register, a secondary status register within the status data structure, to obtain details relating to that event.

Status Data Structures

In general, an asserted STB bit reflects, or summarizes, a change in a corresponding status register. It can have an encoded number which indicates a specific event or condition. Alternately, it can have each of its bits report, or summarize, a change in a third status register, and so on.

Two types of status structures are the Register (individual bits) and the Queue (encoded number) models:

Register Model

The Register Model requires that individual bits identify a specific 7200A condition or event.

Alternately, each bit could act as a summary bit for an associated status register. Using bits in one status register to indicate changes in other registers allows for a layered status description. This layering of detail enables the controller to limit the amount of information it receives. (Note: unless "Register Model" is specifically referenced, the term "register" refers to any byte(s) used to report a status condition or event.)

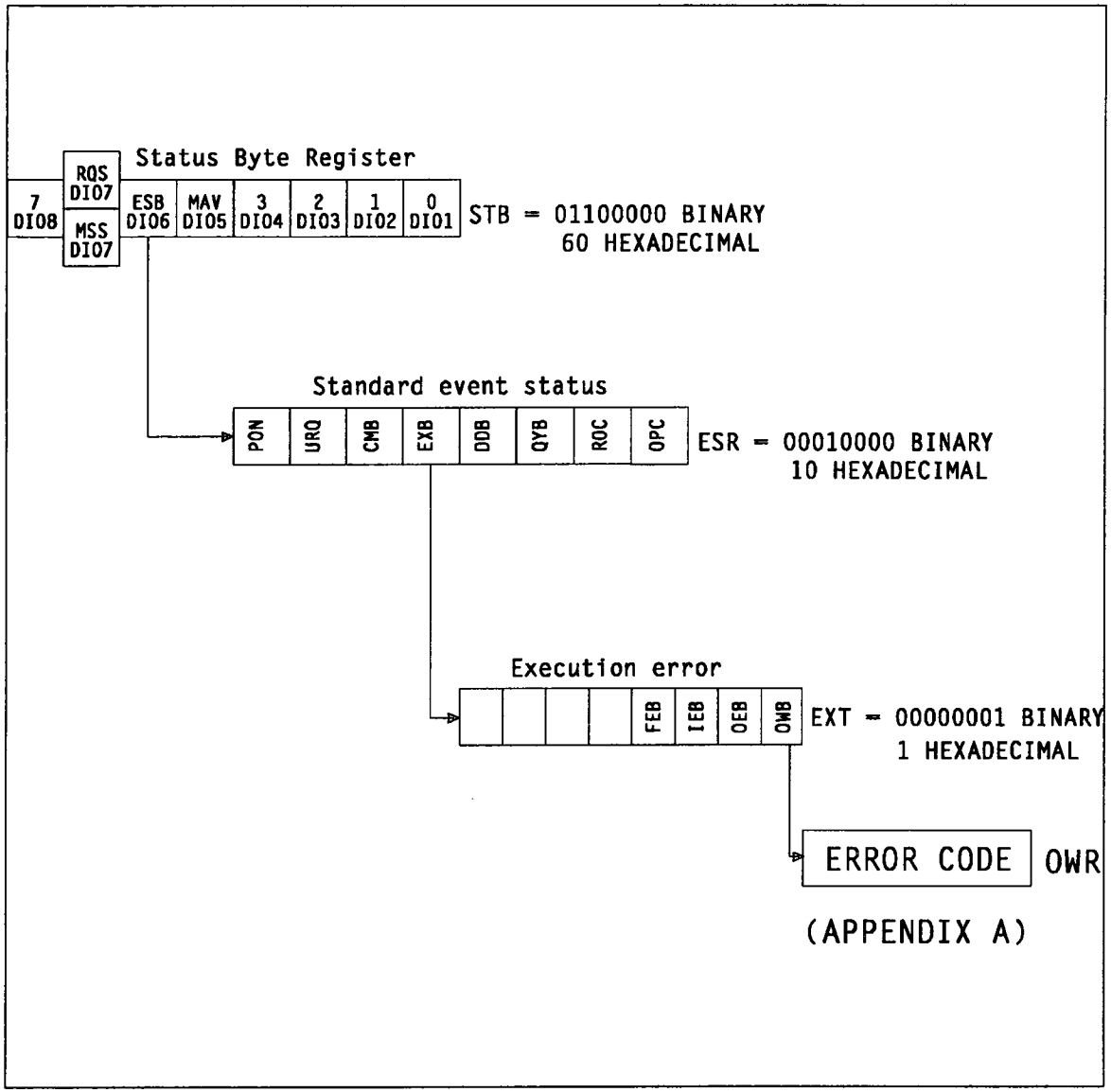
Queue Model

The Queue Model is a single register which contains an encoded number. For example, this number may be an error code which corresponds to an error condition.

The 7200A's queue can hold one error code. When read, the queue will report the most recent error code, and then will clear it.

When the queue is cleared (empty), the corresponding bit in the Register Model will be cleared. Conversely, when the queue contains an error code, the corresponding bit in the Register Model will be set.

The following example illustrates the 7200A's flexible status reporting capability. Each level queried yields more details about the error. Query to the required extent to prevent extraneous information from traversing the GPIB.



4.1 Example of Status Reporting

If you sent the command A:TDIV 20ns to set the timebase of plug-in A to 20 nanoseconds, but the timebase did not change, you could query the STB register using the command

*STB?. The 7200A would return the hexadecimal number 60 (see Figure 4.3), indicating that the Event Status Register (ESR) changed.

Query another level deeper using *ESR?. The 7200A would return the hexadecimal number 10 for the value of the Event Status Register (ESR). This value means the Execution Error bit was set. For more detail send EXR? to read the Execution Error Register. The response of 1 indicates that the Operator Warning bit is set.

Progressing one more step, send OWR? to query the Operator Warning Register. Since this "register" is a queue, the 7200A sends back an error code. Referencing this byte in Appendix A would reveal that a "data out of bounds" error occurred, since plug-in A's timebase cannot go below 50 nanoseconds.

Event Recording

IEEE-488.2 allows two ways to record an event:

Condition Registers — Condition Registers are updated continually and are not cleared when read. The 7200A has no condition registers.

Event Registers — Event Registers capture changes in conditions. They are not cleared until they are read, even if the condition which caused the event no longer exists. Each bit in an Event Register either summarizes a Condition Register, or reports a condition or event in the 7200A.

*NOTE: The *CLS command is provided to clear all Event Registers without reading them first. (The output queue (and its MAV summary bit) is treated as a Condition Register. It is not cleared by the *CLS command.)*

*NOTE: Only changes in bits can propagate to the main status register. Therefore, the *CLS command should be used before monitoring any register. Otherwise, if the bit was previously set, setting it again will have no effect on the main status byte.*

Event Enable Registers

Event enable registers also permit the controller to limit the amount of status information it receives. Every Register Model status register has an associated event enable register. By manipulating the associated event enable register, the controller can selectively enable or suppress (mask) the reporting of specific instrument events. Each bit in an event enable register is "AND'ed" with its corresponding bit in the status register. For example, if the controller sets a bit in the Event Enable register, the corresponding status event can be reported. If it is cleared, the event is masked and its reporting is disabled. In order for a service request to be sent for an event, the corresponding bit in the event enable register, as well as all the bits above it, must be set in order to propagate the bit up to the main status byte. For example, if bit 0 in the *STB register gets set because a trigger got done and its corresponding event enable bit (bit 0) was set in the *SRE register, then the MSS bit will be set. If bit 0 was previously 0, then because it changed to a 1, the RQS bit (bit 6) will be set in the serial poll register (along with bit 0) and the 7200A will generate an SRQ interrupt to the host computer. If the computer performs a serial poll, the 7200A will reset bit 6 after sending its serial poll register to the host computer. However, the *STB register remains unaffected by this and willl still contain decimal 65 (bits 0 and 6 set). Unless the INR register is cleared by reading it or sending *CLS, future trigger completions will not generate an SRQ because the previous state was latched into the INR event register.

If a condition exists such that a bit is set in the *STB register but its corresponding bit in the *SRE event enable register was not set, then the MSS bit and RQS bit will be 0 and no SRQ is generated. However, performing a serial poll will show the bit to be set in the serial poll register. If the event enable bit in the *SRE register then gets set, the unmasked bit set in the *STB register will then set the MSS bit and the RQS bit and generate a service request (SRQ) to the host computer.

In Figure 4.2, when the OWR queue has an error code, the OWR bit is set in the EXR status register.

The OWR bit has a corresponding bit set in the EXR event enable register. Since it is set, the event is reported to the *ESR register. However, this event is not reported to the *STB status register because the corresponding bit in the *ESR event enable register is cleared. Since all other bits are cleared in the *ESR register, the ESB bit in the *STB register is also cleared. Since no bits are set in the *STB , the MSS bit in the *STB is also cleared which causes the RQS bit in the serial poll register to be 0 and no SRQ is generated by the 7200A.

When the 7200A is powered on, all masks are cleared; that is, all status event reporting is disabled.

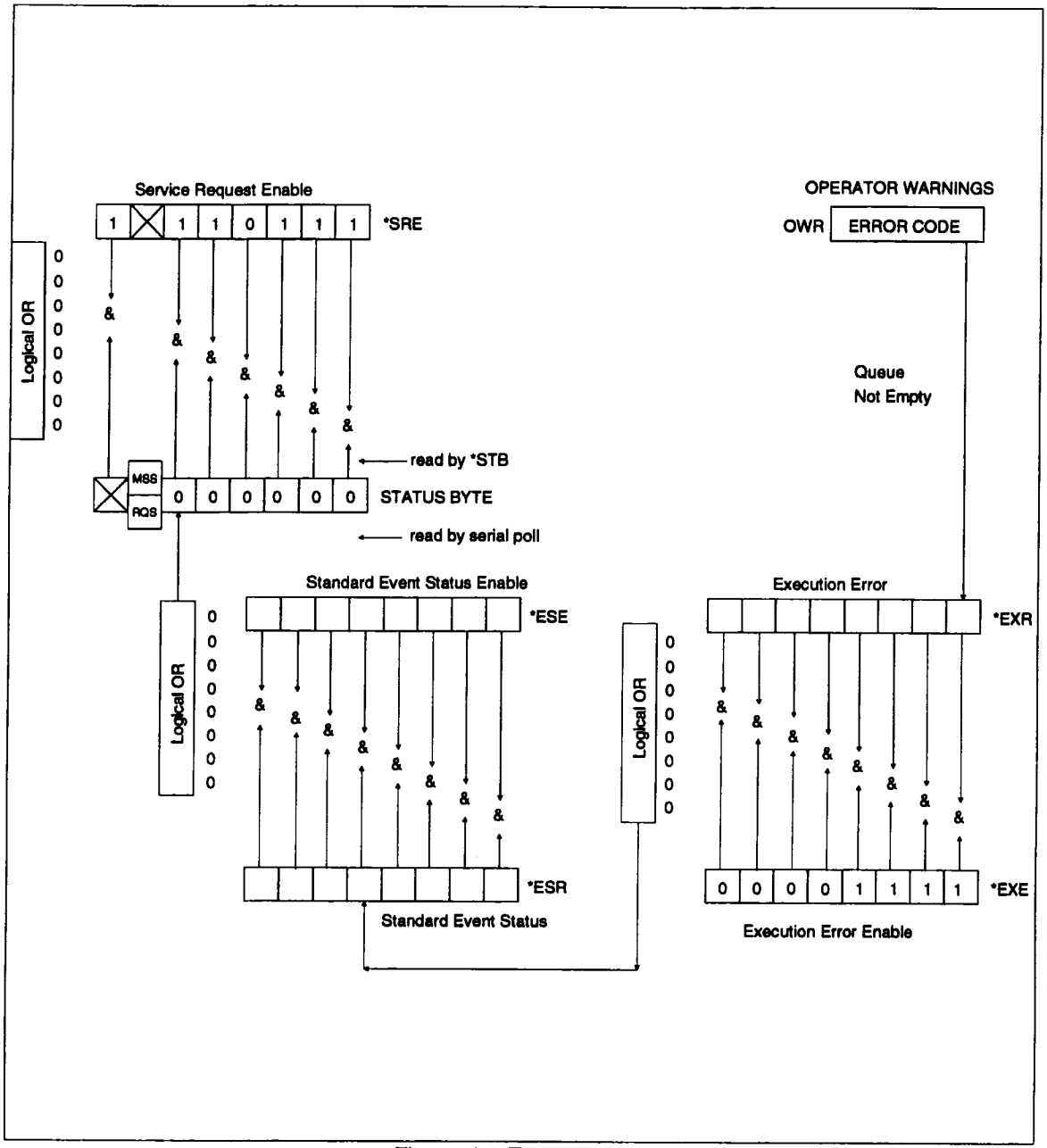


Figure 4.2 Event Enable

Naming Convention

The status registers follow a basic form which can be expressed as "xxR" (i.e., ESR, INR, DPR, ...). Reference to the individual bits within the status register follow the form "xxB" (i.e., ESB, INB, DPB, ...). Reference to the mask used to enable or disable certain bits within each register follow the form "xxE" (i.e., ESE, INE, DPE, ...). The enable register may be set to pass or block the occurrence of certain events using the general form "xxE n" where n is the value to set the enable register. For example, ESE 128 enables the PON bit in the ESR and blocks the other bits. Only registers and masks may be queried according to the general form "xxR?" and "xxE?". The following table summarizes these general forms:

xxB	=	Status xx Bit
xxR	=	Status xx Register
xxE	=	Status xx Enable mask
xxx?	=	Status xxx Query
xxE n	=	Set Status Enable mask xxE to value n

All the status bytes in the 7200A conform to either the register model or the queue model as required by IEEE-488.2. If an event occurs which causes a bit in a register model to change (either from a 0 to 1 or a 1 to 0), that change will be propagated to any overlaying registers, provided the enable mask is set to pass that change. For example, the bits within the ESR register are summarized by the ESB bit in the main status byte register (STB). If a bit in the ESR register changes and the ESE mask is set to pass that change, then the ESB bit in the main status register will be set or reset to reflect the change in the ESR.

If an event occurs which causes a byte to accumulate in a queue model, then the change will propagate to any overlaying registers. For example, the CMR is summarized by the CMB bit in the ESR. If a byte is placed in the CMR queue, the CMB bit will be set to indicate that the queue is not empty. If the ESE mask is set to pass the change in the CMR, then the ESB bit in the main status register will be set.

IEEE-488.2 defines certain "Common Commands", some of which are required for compliance to the standard. All IEEE-488.2 Common Commands are three letter names preceded by an asterisk(*). For example, *RST, *IDN?,...

STB Standard Definition

This section describes the function and structure of all the status registers. The main Status Byte register (STB) reflects instrument status at the time it is read. This register is usually read when the system controller polls the 7200A. Bits in the STB summarize all the other status registers. This summary occurs by "OR'ing" all the reported bits in a summarized register. If the result is TRUE, the summary bit is set. Those registers which the STB summarizes can, in turn, summarize other registers in the same way.

The STB is read with the command *STB?. Its event enable register, or mask, is set with *SRE n. The mask is read with *SRE?. (Note: n is the sum of the decimal bit weights of all bits that are true.)

The *STB? query does not alter any bits in the status byte. Only the *CLS command can clear the status byte, except for the MAV Message Available bit which depends on the state of the Output queue. STB is shown in Figure 4.3.

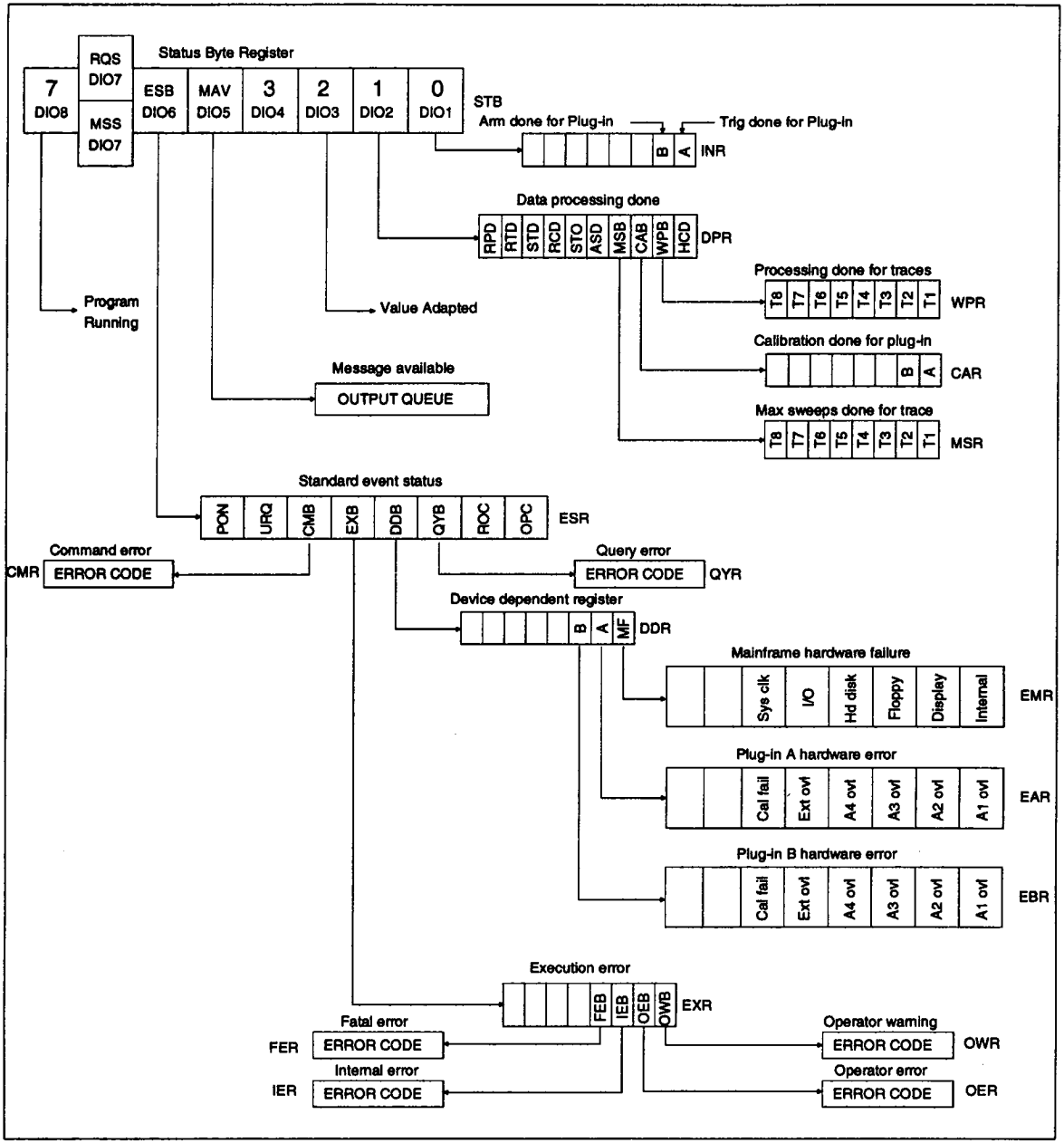


Figure 4.3: 7200A Status Byte Definition

Bit #	Associated Status Byte	Significance
7 (MSB)	none	Program Running
6	none	RQS (service request) Bit
5	ESR	Standard Event Status Bit
4	MAV	Message Available bit
3	none	Reserved
2	none	Value Adapted Bit
1	DPR	Data Processing Bit
0 (LSB)	INR	Internal State Change Bit

Bit 0: INB - Internal State Change Bit

If the INB is set, a plug-in(s) has received a trigger(s). *For Protected Mode operation, other INB bits indicate when a plug-in(s) has been armed.

The INB is a summary of the Internal State Register (INR). INR identifies the plug-in(s) which has received a trigger(s). Since the INR is an event register, any bits stay set until the register is read. After it is read, all the bits are cleared. Once cleared, its summary bit, INB, in the STB is also cleared.

INR's event enable register, or mask, is INE. To set the INE use INE n, and to read it use INE? The command used to read the INR is INR?.

BIT #	Associated Status Byte	Significance
3	none	*Plug-in B armed
2	none	*Plug-in A armed
1	none	Trigger done for plug-in B
0	none	Trigger done for plug-in A

*** for Protected Mode Operation only**

Bit 1: DPB - Data Processing Bit

If the DPB is set, an internal software processing event(s) has completed. The DPB bit summarizes the Data Processing event Register (DPR). DPR identifies which internal software processing event(s) has completed.

Since the DPR is an Event Register, any set bits stay set until the register is read. After it is read, all the bits are cleared. Once cleared, its summary bit, DPB, in the STB is also cleared.

Before waiting for an event in the DPR register, be sure the desired DPR bits are first cleared. Otherwise, a previous event may be read.

DPR's event enable register, or mask, is DPE. To set the DPE use DPE n, and to read it use DPE?

The command to read the DPR is DPR?.

BIT #	Associated Status Byte	Significance
9	none	Replay Traces Done
8	none	Record Traces Done
7	none	Self-test Done
6	none	Recall Done
5	none	Store Done
4	none	Auto Setup Completed
3	MSR	Maximum Sweeps Reached
2	CAR	Calibration Completed
1	WPR	Waveform Processing Completed
0	none	Hardcopy Completed

Note: Bit # 4, Auto Setup Completed, is set when all the plug-ins have been automatically setup. Bit # 1, Waveform Processing Completed, is a summary bit that is set as soon as processing is completed. It is cleared when the WPR register is read.

MSR If the MSB bit (bit # 3 in the DPR) is set, a trace has reached its maximum number of sweeps. The MSR identifies the trace(s) for which the maximum sweeps has been reached. This applies to Summation Averaging, Histograms, Extrema, and any other routines which require a history to be accumulated.

MSR is an Event Register and gets cleared after it is read. MSR's event enable register, or mask, is MSE. To set the MSE use MSE n, and to read it use MSE?. The command to read the MSR is MSR?.

BIT #	Associated Status Byte	Significance
7	none	Max Sweeps reached for Trace 8
6	none	Max Sweeps reached for Trace 7
5	none	Max Sweeps reached for Trace 6
4	none	Max Sweeps reached for Trace 5
3	none	Max Sweeps reached for Trace 4
2	none	Max Sweeps reached for Trace 3
1	none	Max Sweeps reached for Trace 2
0	none	Max Sweeps reached for Trace 1

CAR If the CAB bit (bit # 2 in the DPR) is set, a plug-in(s) has completed calibration. The CAB bit is a summary of the Calibration event register (CAR). CAR identifies which plug-in has completed calibration.

CAR is an Event Register and gets cleared after it is read. CAR's event enable register, or mask, is CAE. To set the CAE use CAE n, and to read it use CAE?. The command to read the CAR is CAR?.

BIT #	Associated Status Byte	Significance
1	none	Calibration Done for plug-in B
0	none	Calibration Done for plug-in A

WPR If the WPB bit (bit # 1 in the DPR) is set, a waveform processing event has occurred. The WPB is a summary of the Waveform Processing Register (WPR). WPR identifies the trace(s) for which processing has completed. For history functions (Average, Histogram, Extrema,...) this event may correspond to a partial result. That is, the waveform processing done bit is set when a trace can be displayed. For history functions, traces can be displayed before the maximum sweeps have accumulated.

WPR is an Event Register that gets cleared after it is read. WPR's event enable register, or mask, is WPE. To set the WPE use WPE n, and to read it use WPE?. The command to read the WPR is WPR?.

BIT #	Associated Status Byte	Significance
7	none	Waveform processing for Trace 8 done
6	none	Waveform processing for Trace 7 done
5	none	Waveform processing for Trace 6 done
4	none	Waveform processing for Trace 5 done
3	none	Waveform processing for Trace 4 done
2	none	Waveform processing for Trace 3 done
1	none	Waveform processing for Trace 2 done
0	none	Waveform processing for Trace 1 done

Bit 2: Value Adapted Bit

The Value Adapted Bit is set to 1 if a received numerical argument was altered before being used in a computation. For example,the 7200A receives "A1:TDIV 11ns". Since the timebase

can only be set in multiples of 1, 2, and 5, the 11ns would get rounded to 10ns. The Value Adapted bit would be set to report that the received value was altered.

Bit 4: MAV - Message Available Bit

MAV is set if data is in the output queue. It informs the system controller that there is still data to output. It is reset once the output queue is empty, indicating that the system controller has read the data from the 7200A. This condition bit is not set or reset when the system controller reads STB. Also, the *CLS command does not affect this bit.

Bit 5: ESB - Event Status Bit

If the ESB is set, an error(s) or user front panel request(s) has occurred. The ESB is a summary of the Event Status Register (*ESR). IEEE-488.2 defines the *ESR to report error conditions common to most automatic test equipment. The 7200A uses most but not all of these bits for synchronization and error reporting.

The *ESR identifies the type of error or whether a front panel request has occurred. Since the *ESR is an Event Register, any set bits stay set until the register is read. After it is read, all the bits are cleared. Once cleared, its summary bit (ESB) in the STB is also cleared.

*ESR's event enable register, or mask, is *ESE. To set the *ESE use *ESE n, and to read it use *ESE?. The command to read the *ESR is *ESR?.

The definition of the bits in the *ESR follow:

BIT #	Associated Status Byte	Significance
7	none	Power On
6	none	User Request(Local Hardcopy Request)
5	CMR	Command Error (syntax,unknown, cmd,...)
4	EXR	Execution Error (invalid parameter,...)
3	DDR	Device Dependent Error (CAL error,...)
2	QYR	Query Error
1	none	Request Control
0	none	Operation Complete

PON This event bit indicates that an off-to-on transition has occurred in the 7200A's power supply.

URQ The User Request bit is set when the Remote Host port and Hardcopy ports are both set to GPIB and the front panel Hardcopy button is pressed. In this case, if the Hardcopy were to start, the 7200A would enter Talk-Only mode and disrupt the connected Remote Host. To prevent this, the User Request bit is set allowing the Remote Host to detect the Hardcopy request and initiate it re-

motely after first setting up all connected devices. Refer to Section 1 for more information.

CMR

If the CMB (Bit # 5 in the ESR) is set, a command parsing error has occurred. The CMB bit summarizes the Command Parsing Register(CMR). The CMR identifies the most recent command parser error. The CMR is a 16-bit queue which contains a unique encoded value. Refer to Appendix A: Command Errors, for a listing of possible command errors.

EXR

If the EXB (Bit # 4 in the ESR) is set, a command execution error(s) has occurred. The EXB bit summarizes the Command Execution register (EXR). The EXR identifies the category of command execution errors. Each of four bits in the EXR corresponds to a different category of error/warning condition. A bit is set when its category of error occurs. A 16-bit queue for each category contains the most recent code.

BIT #	Associated Status Byte	Significance
3	FER	Fatal Error
2	IER	Internal Error
1	OER	Operator Error
0	OWR	Operator Warning

Typical errors include:

- prefix illegal
- too many parameters
- plug-in not present
- data block error
- data block descriptor error
- data processing error
- units do not match

DDR

If the DDB (Bit # 3 in the ESR) is set, a device specific error(s) has occurred. The DDB bit summarizes the Device Dependent Register (DDR). The DDR specifies the origin of the failure (plug-in(s) and/or mainframe). Each bit in the DDR is a summary message bit for a status register corresponding to each plug-in and for the mainframe. The DDR is an event register that gets cleared when read.

DDR's event enable register, or mask, is DDE. To set the DDE use DDE n, and to read it use DDE?. The command to read the DDR is DDR?.

BIT #	Associated Status Byte	Significance
2	EBR	Plug-in B error
1	EAR	Plug-in A error
0	EMR	Mainframe error

ExR Bits These bits indicate the type of device specific error. Each plug-in and mainframe bit in the DDR summarizes an ExR register. The system controller can read these registers to determine the type of error which occurred in the plug-in or mainframe. The definition of the bits in the ExR for all the plug-ins are the same.

The x in ExR represents M,and A and B for mainframe and plug-ins, respectively.

ExR's event enable register, or mask, is ExE. To set the ExE use ExE n, and to read it use ExE?. The command to read the ExR is ExR?.

Typical device specific errors include channel overloads, hardware failures, and self-test failures. This is an event register and gets cleared after it is read.

Bit assignments for ExR, for the plugin:

BIT #	Associated Status Byte	Significance
5	none	Calibration Failed
4	none	External overload
3	none	Channel 4 overload
2	none	Channel 3 overload
1	none	Channel 2 overload
0	none	Channel 1 overload

Bit assignments for EMR:

BIT #	Associated Status Byte	Significance
7	none	reserved
6	none	reserved
5	none	System clock failure
4	none	I/O failure
3	none	Hard disk failure
2	none	Floppy failure
1	none	Display failure
0	none	Internal comm. failure

QYR

If the QYB (Bit # 2 of ESR) is set, a query error(s) has occurred. The QYB bit summarizes the Query Error Register (QYR). QYR identifies the most recent query error. The QYR is a 16-bit queue which contains a unique encoded value. The value indicates query errors. Typical errors include:

Query CHANNEL ALL when both channels are not equal.
Buffer deadlock: input and output buffers full.
Unterminated: controller reads before sending a complete query message.
Interrupted: controller sends new command before reading last query.
Query after indefinite response.

RQC

The Request Control bit in the ESR is never set because the 7200A does not have controller capability.

OPC

To conform to IEEE-488.2, the 7200A will set the OPC bit (Bit # 0 in the ESR) in response to an *OPC command. The query *OPC? will return a "1".

This bit is set upon completion of any operation. At any one time, several operations can be performed and be in various states of completion. To correctly determine which operation completed, monitor the Internal State Register (INR) and the Data Processing Register (DPR).

Bit 6: RQS – Request Service Bit

The RQS bit is a summary bit for the other bits in the STB byte. For GPIB, an SRQ interrupt is generated when the RQS bit is set if the corresponding bit in the SRE mask is set. For RS-232-C, the computer must poll the STB register and read the RQS bit to determine the most recent status.

Bit 7: Program Running Bit

This bit is set when an ICL Program (See Section 7) is executing and is reset when the ICL Program has completed execution.

NOTE: When an ICL Program is running, remote commands are locked out. Therefore, bit 7 of the main status byte should always be checked (using a serial poll) before sending remote commands to ensure proper remote operation.

Section 5: Mainframe Remote Commands

Mainframe Remote Commands (continued)

Mainframe Remote Commands (continued)

Mainframe Remote Commands (continued)

Organization

Each command description begins a new page. A command's name (header) is printed in long and short form near the top of the page. Although the long form is used in the description, the short form can be used instead. Below the header are up to eight sections which describe it:

Purpose:	explains a command's use
Command:	contains a command's syntax
Query:	contains the query syntax
Response:	contains the syntax of the response to a query
Argument:	defines a choice(s)
Example:	includes the command being used
Note:	reports additional considerations
See Also:	cites other relevant commands

Command Execution

Execution of program messages depends on the instrument status. As a rule, commands and queries can be executed in either Local or Remote mode.

Before attempting to execute a command or query, the parser scans it to verify its correctness and that sufficient information is given to perform a requested action.

ALL_STATUS? ALST?

Purpose: Reads the contents of all status registers. After event registers are read, they are cleared. For an interpretation of each register's contents, refer to the appropriate status register description.

The ALST? query is useful for a complete overview of the instrument's state.

Query: **ALL_STATUS?**

Response: ALL_STATUS register,value,register,value,...

Argument:
register Indicates the register mnemonic.
value Indicates register contents.

Examples: **ALL_STATUS?** Read the contents of all status registers.
ALL_STATUS *STB,0,*ESR,0,CMR,0,EXR,0,FER,0,IER,0,OER,0,
OWR,0,DDR,0,EBR,0,EAR,0,EMR,0,QYR,0,
DPR,0,MSR,0,CAR,0,WPR,0,INR,0 < end>

See Also: The query commands for each of these status registers.

ARM_ACQUISITION ARM

Purpose: Enables the signal acquisition process by changing the acquisition state from TRIGGERED to READY.

Command: **ARM_ACQUISITION**

Note: This command works identically as the *TRG command.

See Also: *TRG

AUTO_CAL ACAL

Purpose: To enable or disable automatic calibration of 7200A plugins.

Command: **AUTO_CAL state**

Query: **AUTO_CAL?**

Response: AUTO_CAL state

Argument:

state	ON	Causes the 7200A to periodically perform an automatic recalibration of the plugins.
	OFF	Disables automatic recalibration of plugins.

Examples:

ACAL ON enables automatic calibration.

ACAL? Queries whether automatic calibration is enabled or disabled.

ACAL ON Reports than automatic calibration is enabled.

Note: Automatic calibration is always enabled when the 7200A is powered on or reset.

*When controlling the 7200A from remote, it is recommended that automatic calibrations are disabled, since they will occur asynchronously and can take several seconds per plug-in to complete. However, if you do disable it, periodic calibrations should be performed via the *CAL? command.*

See Also: *CAL?

AUTO_SETUP ASET

Purpose: Sets up acquisition parameters automatically. The command attempts to display the input signal(s) by adjusting the vertical, timebase, and trigger parameters. If input signals are available and more than one channel is being displayed, the signals will be scaled vertically and the timebase will be chosen to best display Channel 1. If only one input channel is turned on, the timebase will be adjusted for that channel.

Command: **AUTO_SETUP**

Notes: When locked timebase is selected, the timebase controls will be set up based on the leftmost plugin's control settings.

AXIS_LABEL AXIL

Purpose: This command enable/disables the display of the end point values for the active trace on both the horizontal and vertical axes. If enabled, the values are printed when the grid is either single or dual.

Command: **AXIS_LABEL state**

Query: **AXIS_LABEL?**

Response: AXIS_LABEL state

Arguments:

state	ON	displays the values corresponding to the active trace at the end of each axis.
	OFF	takes the values of the axes off the display.

Example:

AXIL	ON	tells the 7200A to display the axis values.
AXIL?		asks if the axis lables are on or off.
AXIL	OFF	response indicates axis labels are not displayed.

See Also: SELECT

BUZZER BUZZ

Purpose: Briefly sounds the buzzer.

Command: **BUZZER**

CAE

Purpose: Sets the Calibration Enable register (CAE). The CAE register determines which events in the Calibration status Register (CAR) are reported. CAR identifies which plug-in has completed calibration. Any reported CAR event sets the CAB summary message bit (bit # 2) of the Data Processing Register (DPR) and propagates to the main Status Byte (STB).

Command: **CAE mask**

Query: **CAE?**

Response: CAE mask

Argument: mask When expressed in binary, this number (between 0 and 63) represents the bits of the CAR that can be reported:

Bit #	Associated Significance
1	Calibration Done for plug-in B
0	Calibration Done for plug-in A

Examples: **CAE 3** enables the events represented by the lower two bits of this event register (i.e., calibration done for plug-ins A and B). bit 1 (decimal 2), and bit 0 (decimal 1). Summing the decimal values yields 3.

CAE? Read the contents of the CAE.
CAE 3 Response indicates the contents as 3, i.e., the lower two enable bits are set.

See Also: CAR?

CAR?

Purpose: Reads and then clears the Calibration event Register (CAR) which identifies which plug-in has completed calibration.

Clearing the CAR register also clears the CAB summary message bit (bit # 2) of the Data Processing Register (DPR) and the effect could propagate to the main Status Byte (STB).

Query: **CAR?**

Response: CAR value

Argument value When expressed in binary, this number (between 0 and 63) represents the bits of the CAR:

Bit #	Associated Significance
1	Calibration Done for plug-in B
0	Calibration Done for plug-in A

Examples:

CAR? Read and clear the CAR register contents.

CAR 2 Response indicates that calibration done for plug-in B.

See Also: ALL_STATUS, CAE, *STB?

CENT

Purpose: Reads and writes the centronics port for synchronization and control of external events. The command and query can be used in a variety of ways such as limit testing and external triggering. Since the 8 outputs provide standard TTL levels, they can be used to initiate some external action while the 7200A is left monitoring (babysitting) an input signal.

Command: **CENT value**

Query: **CENT?**

Response: CENT value

Arguments: value The actual data to be written to or read from the centronics port. When writing, value may be specified in decimal, hexadecimal (# H), octal (# Q), or binary (# B).

When writing the centronics port, the actual pins which correspond to the data value are as follows:

Data Bit	DB25-D pin	36-pin bail lock
D0	2	2
D1	3	3
D2	4	4
D3	5	5
D4	6	6
D5	7	7
D6	8	8
D7	9	9

CENT (continued)

When reading the centronics port using CENT?, the data value returned is bit-encoded as follows:

Data Bit	DB25-D pin	36-pin bail lock	Centronics bail lock Signal Name
D0	10	10	ACKNOWLEDGE
D1	11	11	BUSY
D2	12	12	PAPER OUT
D3	13	13	SELECT
D4	15	32	ERROR

Examples:

CENT 8 Sets pin 5 high on the DB25-D Centronics parallel output connector; pins 2,3,4,6 through 9 are set low.

CENT # B11001010 Sets pins 3,5,8 and 9 high and pins 2,4,6 and 7 are set low on the DB25-D connector.

CENT?
CENT 21 Response indicates that pins 10,12, and 15 are pulled high and pins 11 and 13 are pulled low on the DB25-D connector (decimal 21 = 00010101 in binary)

Note:

The CENT command is not intended to drive a printer since the normal data transfer sequence is not carried out.

Data written to the DB25-D connector remains latched on pins 2 through 9 until another byte overwrites it.

All voltages appearing on pins 2 through 9 are based on TTL levels and are driven by a continuously enabled bus transceiver. When interfacing external devices, please refer to the appropriate data sheet to avoid overloading.

The CENT? query allows external hardware signals to be read remotely. If there is no connection to these input pins, they are considered to be floating and the query response may return random data. If only some input pins are used, then the appropriate data bits should be isolated from the query response.

CENTER_MAX CMAX

Purpose: Automatically moves the center of a histogram to the mode of the currently accumilated histogram.

Command: **prefix:CENTER_MAX**

Arguments:

prefix	The prefix is limited to traces T1, T2,..., T8

Examples:

T3:CMAX	Automatically moves the center of the histogram defined in trace 3 to the mode of the current histogram.

See Also: FIND_CTR_RANGE

CLEAR_DISPLAY CLRD

Purpose: Reset the data associated with the history function(s) as applicable (e.g., Average, Extrema, etc.) and also clears the persistence display.

Command: **CLEAR_DISPLAY**

Note: When this command is received, all data generated by the history functions (Average, Extrema, Histogram, and Trend) is set to zero and the count(s) of the number of sweeps is set to zero. Also, if persistence is on, the display is cleared and its sweep count is reset to zero. If no history function is being used and persistence is not on, this command has no effect.

See Also: DEFINE, DEFINE_REPLAY, PERSIST

CLEAR_MEMORIES CLRM

Purpose: Clears the eight memories M1 ...,M8 and the record traces buffer.

Command: **Clear_Memories**

CMR?

Purpose: Reads and then clears the contents of the Command error Register (CMR). The CMR register identifies the most recent command parser error. A command error is reported whenever the 7200A detects a syntax error or is unable to parse the command or query.

The CMR is a single-value queue which contains a unique encoded value. The value corresponds to the most recent command error. An encoded value is assigned to every error message that can appear on the screen. Values are also assigned to remote specific errors. A listing of all encoded values with their meanings is found in Appendix A.

Clearing the CMR register also clears the CMB summary message bit (bit # 5) of the standard Event Status Register (ESR) and the effect could propagate to the main Status Byte (STB).

Query: **CMR?**

Response: CMR value

Argument:

value	Corresponds to an error. Typical errors include: Header error String error Keyword error Number error Suffix error Prefix illegal Too many parameters EOI detected during definite length data block transfer

Example:

CMR?	Read and clear the CMR.
CMR 453	Response indicates unknown remote command.

See Also: ALST?, *CLS, *STB?

COLOR COLR

Purpose: Determines the color assignments for the different display entities as well as how many color should be used to display the traces.

Command: **COLOR keyword, value [,keyword, value]**

Query: **COLOR?**

Response: COLOR keyword, value, keyword, value, keyword, value

Argument: An argument consists of a keyword followed by its value. Any number of arguments, in any order may be used.

Keyword	Meaning: Value
TRACES	Selects how many different colors should be used to display the traces. Choices are 1 (all traces the same color), 2 (the active trace one color, all other traces a second color), and 8 (each trace is a different color).
SCHEME	Selects the color scheme to use. The color scheme is a file (in the panel setup directory with a ".COL" extension for color systems and ".MON" for monochrome systems) that contains a list of the color assignments for each possible display entity. The file DEFAULT.COL contains a description of each of the fields.
CONTRAST	Selects the difference in intensity of the dark, medium, and bright colors of each hue. Bright colors are used for highlighting entries which you can change in setup screens and for acquired points in expanded waveforms. The contrast is a percentage ranging from 5, in which there is very little difference in intensities, to 95, where all but the bright colors are nearly black.

Example:

COLR SCHEME, DEFAULT, TRACES, 8
Sets the color scheme according to the file "DEFAULT" and specifies that a different color should be used for each trace.

COLR? requests the current color settings

COLOR (continued) COLR

COLR SCHEME, PRIMARY, TRACES, 2, CONTRAST, 30
reports that the color scheme is currently determined by the file "PRIMARY" and that only the active trace is a different color, and the contrast is 30.

Note: For the 7200A with the color option, the ICL program, COLOR, can be used to generate your own customized color scheme.

See Also: INTENSITY

COMM_FORMAT CFMT

Purpose: Determines the format which the 7200A will use to send waveform data. The available options set: (1) the data block format, (2) the data word size, and (3) the data encoding to be modified from the default settings

Command: **COMM_FORMAT format, data_width, encoding**

Query: **COMM_FORMAT?**

Response: COMM_FORMAT format, data_width, encoding

Arguments: format, choice of data block format:

DEF9 definite length, has the block format :
9nnnnnnnnn< DB1> < DB2> ...< DBn>

IND0 indefinite length, has the block format:
0< DB1> < DB2> ...< DBn> < end>

OFF has the block format:
< DB1> < DB2> ...< DBn> < end>

where nnnnnnnnn is a decimal integer defined by nine bytes and indicates the number of data bytes. < DBn> represents a data byte and < end> is the message terminator.

data_width, choice of:

BYTE	1 byte
WORD	2 bytes

encoding, choice of:

BINary	the most compact, provides the fastest transfer
HEXadecimal	two ASCII characters, 0" through 9" and A through F, represent the contents of each data byte. The first character represents the four most significant bits.

COMM_FORMAT (continued) CFMT

Examples:

CFMT DEF9,WORD,BIN Sets the format to # 9 definite length block, with 16 bit data words in binary encoding.

CFMT?
CFMT IND0,BYTE,HEX
Requests the current format settings
Reports the data block is indefinite length format, with 8 bit data words in hexadecimal encoding.

Note:

RS-232-C requires HEX encoding, GPIB can use either.

Waveform data points result from 8-bit ADCs so that a data_width of BYTE can fully describe each point. Selecting a data_width of WORD simply places the BYTE in the upper half of a 16-bit WORD and clears the lower half.

The format set by this command only affects the WAVEFORM command.

The format OFF is an extension to the IEEE-488.2 standard and is provided for special applications where the absolute minimum of data transfer may be important. It suppresses any keywords normally included in the response, and comma separators.

See Also:

WAVEFORM

COMM_HEADER CHDR

Purpose: Specifies which type of ASCII header, if any, should be sent with a response to a query. The response header can have three forms:

LONG:	Full command name followed by its argument(s),
SHORT:	Abbreviation of the command followed by its argument(s), or
OFF:	Just the argument(s) with no command name.

Command: **COMM_HEADER arg**

Query: **COMM_HEADER?**

Response: COMM_HEADER arg

Arguments: arg, choice of :
OFF (no header),
SHORT (abbreviated command name), or
LONG (full command name)

Examples:

CHDR SHORT	Tells the 7200A to start responses with the abbreviated command names.
CHDR?	Requests the current header type.
COMM_HEADER LONG	Reports the current header type as LONG.

If the 7200A received T1:VPOS?, it would respond with:

T1:VERT_POSITION 1.0	if CHDR LONG
T1:VPOS 1.0	if CHDR SHORT
1.0	if CHDR OFF

Note: COMM_HEADER only affects the response header. Long or short forms can always be used for sending commands to the 7200A. Arguments are not affected.

COMM_ORDER CORD

Purpose: Specifies the order of bytes for two byte data words. High byte (most significant byte) first is generally known as Motorola format. Low byte (least significant) first is called Intel format and is also used by an IBM PC or compatible.

Command: **COMM_ORDER arg**

Query: **COMM_ORDER?**

Response: COMM_ORDER arg

Argument: arg choice of HI or LO

Examples:

CORD LO	Causes the least significant byte of a two byte data word to be sent first. (This is how it should be set when reading into an IBM PC or compatible.)
CORD?	Requests the current order in which bytes are sent.
CORD HI	Indicates Motorola format

Note: COMM_ORDER only applies to block data in the WAVEFORM command.

See Also: WAVEFORM

COMM_RS232 CORS

Purpose: Defines the RS-232-C message exchange protocol.

Command: **COMM_RS232 keyword,value, keyword,value,...,keyword,value**

Query: **COMM_RS232?**

Response: COMM_RS232 keyword,value,keyword,value

Arguments:

keyword	Meaning: values	Initially selected value
EI	End_In terminates command messages: 1 through 127	13
EO	End_out terminates response messages: Alphanumeric string of no more than two characters	\r\n
LS	Line_separator: CR, LF, CRLF, or OFF	OFF
LL	Line_length, for LS not OFF: 40 through 1024	1024
EC	Echo determines if received characters are sent back	OFF
BR	Baud Rate: 19200, 9600, 4800, 2400, 2000, 1800, 1200, 1050, 600, 300, 200, 150, and 110	
SB	Stop Bits: 1 or 2	
PR	Parity: NONE, ODD, or EVEN	
DB	Data Bits: 5, 6, 7, or 8	
HS	Handshake method: HARDWIRE, XON-XOFF	

An argument consists of a keyword followed by its value. Any number of arguments may be used in any order to change individual settings.

Examples: **CORS EC,OFF,LL,80** Selects echo off and a line length of 80.

COMM_RS232 (continued) CORS

CORS? Query, which has no parameters, requests the state of all settings.

CORS BR,9600,SB,1,PR,NONE,DB,8,HS,XON-XOFF,EI,13,EO,"\ r\ n", LL,1024,LS,OFF,EC,OFF

Note: The keywords have the following meanings:

Echo (EC)	Determines if received characters are sent back to the remote host. ON indicates FULL duplex mode, OFF indicates HALF duplex mode.
End_In (EI)	Terminates the command message sent to the 7200A. Be sure to use a value that is not included in the message itself, otherwise the 7200A will interpret it as the terminator.
End_Out (EO)	Terminates the response message sent from the 7200A. One or two characters are used to represent the string. Unix-like notation is used to represent control characters. For example, \ r\ n is used to mean CR (decimal 13) followed by LF (decimal 10)
Line_Separator (LS)	Terminates a line of response, particularly useful for formatting a printout of block data. If OFF is selected, Line_Separator is not used.
Line_Length (LL)	Sets the maximum number of characters per line. If the Line_Separator occurs before the Line_Length is reached, the line is terminated.

Whenever the Line_Separator character(s) is encountered, the count for the number of characters on a line is reset. Whenever the count exceeds Line_Length, the Line_Separator is transmitted before the current character.

See Also: Block Data, Command Syntax, WAVEFORM

COMM_SCSI COSC

Purpose: Sets or determines the SCSI id and block size to be used in transfers of non-corrected acquisition data.

Command: **COSC keyword, value[,keyword,value]**

Query: **COSC? keyword[,keyword]**

Response: COSC keyword,value[,keyword,value]

Arguments:

ID: Specifies the SCSI id of the device to which the acquired data are to be sent. Choices are 0 to 7, excepting the SCSI id of the 7200A itself.

BS: Specifies the number of bytes in each block to be transferred. The last block may be smaller than the specifies block size, depending on the amount of data acquired. The block size can range from 4 to 65536 in increments of 2.

Example:

COSC ID, 6, BS, 2048	Sets ID= 6, block size= 2k
COSC?	Requests the current settings
COSC ID,6, BS, 2048	Returns current settings

COPY_FILE COPY

Purpose: Copies the specified file(s) from the internal disk to the floppy or vice versa.

Command: **COPY_FILE keyword,value**

Argument: The argument consists of a keyword followed by its value.

Keyword	Meaning: Value
INPN	Panel settings file on the internal disk to be copied to the floppy disk. "filename.ext" filename including extension or "ALL FILES"
INAS	Program file on the internal disk to be copied to the floppy disk. "filename.ext" filename including extension or "ALL FILES"
FLPY	Panel settings or program file on the floppy disk to be copied to the appropriate directory on the internal disk. "filename.ext" filename including extension or "ALL FILES"

Examples:

COPY FLPY, "MIKE.PNL"	Copy the panel settings file MIKE.PNL from the floppy to the internal disk.
COPY INAS, "ALL FILES"	Copy all program files from the internal disk to the floppy disk

See Also: DIRECTORY_LIST, DELETE_FILE

CURSOR_LOCK CRLK

Purpose: Allows you to choose to have independent parameter cursors for each trace or have all the cursors locked together so that they are on the same place on all the traces.

Command: **CURSOR_LOCK argument**

Query: **CURSOR_LOCK?**

Response: CURSOR_LOCK argument

Arguments: The argument specifies whether to turn CURSOR_LOCK on or off:

ON	Provides one parameter cursor for all of the traces.
OFF	Provides an independent parameter cursor for each trace.

Example:

CURSOR_LOCK ON	Selects one parameter cursor for all of the traces.
CURSOR_LOCK?	Returns the current status of of CURSOR_LOCK .
CURSOR_LOCK OFF	Indicates cursor parameters may be set independently for each trace.

Notes: When using Histogram, you may want the cursors for all traces to be independent of each other. The reason for this is that if the cursors were locked and you moved the cursors on the histogram, you may also be moving the cursors on the trace which is providing the input to the histogram and will therefore cause the histogram to be reset.

See Also: CURSOR_SET, PARAMETER_VALUE

CURSOR_MEASURE CRMS

Purpose: Displays the specified cursor type if the cursors are not already on.

Command: **CURSOR_MEASURE keyword**

Query: **CURSOR_MEASURE?**

Response: CURSOR_MEASURE keyword

Argument:

keyword	meaning	
VREL	Vertical Relative	Measures the difference between the vertical positions of two cursors.
VABS	Vertical Absolute	Measures the absolute vertical value at a given point.
HABS	Marker	Measures the absolute horizontal position and its vertical value of a point on a trace(s).
HREL	Horizontal	Measures the difference between the horizontal positions and the corresponding vertical values of two Horizontal cursors.
PARAM	Basic Parameter	Calculates a fixed set of waveform parameters on one trace between two cursors.
EXPAR	Extended	Calculates up to 20 user selected parameters that can be defined on any combination of traces.
OFF		Display no cursors.

Examples:

CRMS? Requests which cursors are currently displayed.
CRMS VREL Reports that only the Vertical cursors are being displayed.

Note: This command affects the display of cursors, it does not affect the positioning of cursors (see CURSOR_SET) or their measurement (see CURSOR_VALUE).

See Also: CURSOR_SET, CURSOR_VALUE, PARAMETER_VALUE, XY_CURSOR_SET, XY_CURSOR_VALUE, PER_CURSOR_SET, PER_CURSOR_VALUE

CURSOR_SET CRST

Purpose: Positions any one of the eight independent cursors at a given screen location when not in XY or persistence mode. Cursor positions are specified relative to a grid. The positions of the cursors can be modified or queried even if the required cursor is not currently displayed on the screen.

Command: **prefix:CURSOR_SET keyword,position,...,keyword,position**

Query: **prefix:CURSOR_SET? keyword,keyword,keyword,...**

Response: prefix:CURSOR_SET keyword,position,...,keyword,position

Arguments: prefix The prefix is limited to traces, i.e., T1, T2, ..., T8.

keyword,position,...,keyword, position

Cursor Type	keyword	position
Vertical Absolute	VABS	-4 to 4 DIV
Vertical Relative	VREF, VDIF	-4 to 4 DIV
Marker	HABS	0 to 10 DIV, or Horizontal units
Horizontal	HREF, HDIF	0 to 10 DIV
Basic	PREF, PDIF	0 to 10 DIV
Extended	PREF, PDIF	0 to 10 DIV
Unlocked	UREF, UDIF	0 to 10 DIV

The seven cursor types measure the following:

Verticial Absolute	measures the absolute vertical value at a given point.
Vertical Relative	measures the difference between vertical positions of the cursors of a trace(s).
Marker	measures the absolute horizontal position and its vertical value of a point on a trace(s).
Horizontal	measures the difference between the horizontal positions and their corresponding vertical values of two Horizontal cursors
Basic	calculates a fixed set of waveform parameters on one trace between two cursors

CURSOR_SET (continued) CRST

	Extended	calculates up to 20 user selected waveform parameters that can be defined on any combination of traces.
	Unlocked	Same as basic & Extended cursors but can be positioned individually on each trace

Examples:

T1:CRST HABS, 5	Places the marker in the center of the waveform.
T1:CRST? HREF	Requests the position of the horizontal reference cursor on trace 1
T1:CRST HREF, 2	The response indicates the position is two divisions to the right of the grid's left edge.

Note:

Keywords ending in REF refer to a reference cursor which is a line of alternating dots and dashes. Keywords ending in DIF correspond to the difference cursor which contains dashes.

When positioning VREF and VDIF cursors and when the 7200A is displaying more than one grid, the specified position is interpreted as the number of divisions above or below the center of the grid on which the trace is currently displayed.

A command argument has two parts: keyword followed by position. Any number of arguments may be used in any order to change individual settings.

If no arguments are specified with the query, all cursor positions are returned.

When querying the position, if the cursor is not on the specified trace, the value UNDEF is returned.

See Also:

CURSOR_MEASURE, CURSOR_VALUE, PARAMETER_VALUE, PER_CURSOR_SET, XY_CURSOR_SET

CURSOR_VALUE? CRVA?

Purpose: Returns the values of the specified cursor measurement(s) for a given trace when not in XY or persistence mode. The corresponding units accompany each reported value.

Query: **prefix:CURSOR_VALUE? keyword,keyword,...,keyword**

Response: prefix:CURSOR_VALUE keyword,value,value,keyword,value,...

Arguments: prefix The prefix is limited to traces, i.e., T1, T2, ... T8.

keyword,...,keyword

keyword		meaning
VABS	Vertical Absolute	measures the absolute vertical value at a given point
VREL	Vertical Relative	measures the difference between the vertical positions of two cursors
HABS	Marker	measures the absolute horizontal position and its vertical value of a point on a trace(s).
HREL	Horizontal	measures the difference between the horizontal positions and their corresponding vertical values of two horizontal cursors

value indicates the cursor measurement results with the correct units.

Examples: **T1:CRVA? HREL** Requests the horizontal and vertical difference(s) between the co-ordinates of the two Horizontal cursors

T1:CRVA HREL,21.3 MS, 5.1 DB

T1:CRVA? VABS Requests the measurement for the Vertical absolute cursor position.

T1:CRVA VABS, -24 mV

Notes: If no keywords are specified, the values of all measurements are returned.

For Marker and Horizontal cursors, two values, vertical and horizontal, are returned. For vertical and marker cursors, one value is returned.

CURSOR_VALUE? (continued) CRVA?

If the trace has multiple arrays, such as, EXTREMA or a Complex FFT, the marker and horizontal cursors return a second vertical value.

Any number of cursor types can be specified in the argument. Their cursor names followed by the value(s) and units are returned in the same order as requested.

To measure a cursor value, its cursor type need not be selected by the CURSOR_MEASURE command. Use PARAMETER_VALUE? to read Basic and Extended cursor parameter values.

If the cursor is not on the specified trace or the trace is not valid, the value UNDEF is returned.

See Also: CURSOR_MEASURE, CURSOR_SET, PARAMETER_VALUE? PER_CURSOR_VALUE, XY_CURSOR_VALUE

DATA_DEST (Available for Option F2 Only) DDST

Purpose: Selects the destination for data from all of the plugins when sequence mode is active.

Command: **DATA_DEST value**

Query: **DATA_DEST?**

Response: DATA_DEST value

Arguments:

value STANDARD When in sequence mode, segments are collected in the plugin, processed and displayed locally.

SCSI When sequence mode is active, the data acquired is not accessible to the 7200A for any internal processing or display; it merely passes through from the plugin to the SCSI port. Therefore, the display of all traces is automatically disabled and no computational processes can be applied to the data.

Examples:

DATA_DEST SCSI	sets the data destination for sequence mode to the SCSI port.
DATA_DEST?	Queries the current sequence mode data destination
DATA_DEST STANDARD	Indicates that sequence mode data will be processed and displayed locally.

See also: SCSI_ID, COMM_SCSI, TRIG_MODE

DATE

Purpose: Sets the date and time of the 7200A's internal real time clock.

Command: **DATE** day,month,year,hour,minute,second

Query: **DATE?**

Response: DATE day,month,year,hour,minute,second

Arguments:

day	1 through 31
month	First three letters of the month's name: JAN, FEB, MAR, APR, MAY, JUN, JUL, AUG, SEP, OCT, NOV, DEC
year	Valid years: from 1989 through 2050.
hour	24-hour notation: 0 (midnight) through 23 (11 p.m.).
minute	0 through 59
second	0 through 59

Examples:

DATE 27,JUL,1989,10,12,32
Changes the date to July 27, 1989 and the time to 10:12:32 in the morning.

DATE?	Query returns the date and time in the same format used to set the date and time.
DATE 12,AUG,1989,16,10,59	Returns the current day, month, year, hour, minute, and second.

DDE

Purpose: Sets the Device Dependent Error enable register (DDE). The DDE register determines which events in the Device Dependent Register (DDR) are reported. DDR identifies the plug-in which caused the device specific error. A mainframe error is also reported in the DDR. Any reported DDR event sets the DDB summary message bit (bit # 3) of the standard Event Status Register (ESR) and propagates to the main Status Byte (STB).

Command: **DDE mask**

Query: **DDE?**

Response: DDE mask

Argument: mask — When expressed in binary, this number (between 0 and 127) represents the bits of the DDR that can be reported:

Bit #	Significance
2	Plug-in B error
1	Plug-in A error
0	Mainframe error

Examples:

DDE 7 — Enables the events represented by the lower three bits of this event register (i.e., mainframe error and plug-ins A, and B)
Set bit 2 (decimal 4), bit 1 (decimal 2), and
bit 0 (decimal 1). Summing the decimal values yields 7.

DDE? — Read the contents of the DDE.
DDE 7 — Response indicates the contents as 7, i.e., the lower three enable bits are set.

See Also: EME, EAE, EBE

DDR?

Purpose: Reads and then clears the contents of the Device Dependent Register (DDR). In the case of a hardware fault, the DDR register specifies the origin of the fault (plug-in(s) and/or mainframe).

Each bit in the DDR is a summary message bit for a status register corresponding to each plug-in and for the mainframe. The DDR will not clear a bit if its corresponding register is not cleared.

Clearing the DDR register also clears the DDB summary message bit (bit # 3) of the standard Event Status Register (ESR) and the effect could propagate to the main Status Byte (STB).

Query: **DDR?**

Response: DDR value

Argument: value When expressed in binary, this number (between 0 and 127) represents the bits of the DDR.

The status bytes that each DDR bit summarizes and the bit's assignment follow:

Bit #	Associated Status Byte	Significance
2	EBR	Plug-in B error
1	EAR	Plug-in A error
0	EMR	Mainframe error

Example

DDR? Read and clear DDR register contents.
DDR 3 Response indicates that mainframe and plug-in A errors are reported.

See Also: ALST?, DDE, EAR?, EBR?, EMR?, *STB?

DEFINE DEF

Purpose: Defines a trace equation, which specifies the source(s) of data and how it is processed.

Command: **prefix: DEFINE keyword,value...,keyword,value**

Query: **prefix: DEFINE?**

Response: prefix: DEFINE keyword,value...,keyword,value

Arguments: prefix The prefix for this command is limited to traces, i.e., T1, T2, ... T8.

One or more of the following keywords may be sent, in any order, with each followed immediately by its value:

keyword	applies to	meaning:value
EQN	all	the right side of the equation for the trace, in double quotes
MAXPTS	all	the maximum number of points to produce in the trace: 50,100,200,500,1000,2000,5000, 10000,20000,50000
SWEEPS	AVGS EXTREMA	the maximum number of sweeps to accumulate: 10,20,50,100,200,500,1000...1000000
REJECT	AVGS	turns artifact reject (rejection of waveforms with underflows or overflows): ON or OFF
WEIGHT	AVGC	weighting factor: 2,4,8,16,32,64,128,256
RES	ERES	nominal number of effective bits after enhancement: 8.0,8.5,9.0,9.5,...11.0
WINDOW	FFTIM,FFTMAG, FFTPHA,FFTPWD FFTPWS,FFTRE,	the FFT window type: VON_HANN,HAMMING BLKMN_HARRIS, RECTANGULAR,FLAT_TOP, EXPONENTIAL

DEFINE (continued) DEF

keyword	applies to	meaning;value
DCSUP	FFTIM,FFTMAG, FFTPHA,FFTPWD FFTPWS,FFTRE FTRI	turns DC suppression (removal of DC mean of input waveform before application of FFT): ON or OFF
MAXBINS	HIST	the number of bins in the histogram: 20,50,100,200,500,1000,2000
PARAM	HIST,TREND	the parameter of the source waveform whose value is to be histogrammed or trended The available parameters are listed in the description of the PAAD command on pp. 5-74 Arguments to parameters are specified as for PAAD.
MAX_EVENTS	HIST	the maximum number of parameter value events to accumulate in the histogram: 20,50,100,200,500,1000,..2000000000
CENTER	HIST,TREND	midpoint of the range of values of a histogram or trend: -1E30 to 1E30
WIDTH	HIST	horizontal scale per division of a histogram i.e. 1/10 of the total histogram width): 1E-30 to 500E30
WIDTH	LIMIT	The amount of flattening in the horizontal direction, specified in divisions.
HEIGHT	TREND	vertical scale per division of a trend i.e, 1/8 of the total range of values trended): 1E-30 to 500E30
HEIGHT	LIMIT	The amount of flattening in the vertical direction, specified in divisions.
VERT	HIS	vertical scaling of a histogram:

DEFINE (continued) DEF

		LIN,LOG,MAX
HORIZ	TREND	horizontal scaling of a trend: EVENTS, 1,2,5,10,20,50,...5000 numerical values specify time per division
EVENTS_PER_WF	HIST TREND	the number of parameter values which are accumulated in a histogram or trend for each source waveform sweep FIRST,ALL,AVERAGE
HORIZ	TREND	horizontal scaling of a trend: EVENTS, 1,2,5,10,20,50,...5000 numerical values pecify time per division
EVENTS_PER_WF	HIST TREND	the number of parameter values which are accumulated in a histogram or trend for each source waveform sweep FIRST, ALL, AVERAGE
SLOPE	TOFPC, TOFAN	Specifies whether rising or falling edges are used to compute peak times. Choices are POS and NEG.
COUPLING	TOFPC	indicates the method of applying the Threshold for locating edges. Choices are AC and DC.
THRESHOLD	TOFPC	Indicates the value for computing the time of an edge. The value is expressed in the vertical units of the input waveform. If Coupling is DC, the Threshold is an absolute value. If Coupling is AC, the Threshold is relative to the baseline of the input:-1E30 to 1E30.
GAIN	TOFAN, TOFPC	used to define the mass axis:-1E30 to 1E30.
OFFSET	TOFAN, TOFPC	used to define the mass axis:-1E30 to 1E30.
SPACING	WINHIST	distance from the center of one window to the center of the next window in the horizontal units of the input waveform: 1E30 to 1E30

DEFINE (continued) DEF

The default equations for the eight traces use the DEFINE (continued)first eight input channels
starting with those in plug-in slot A. If less than eight channels are available, the list will repeat. For example, if there are two plug-ins, A and B, with three channels each, the equations will be the following:

T1 = A1 T2 = A2 T3 = A3 T4 = B1
T5 = B2 T6 = B3 T7 = A1 T8 = A2

Examples:

T4:DEFINE EQN, DIFF((30MV*A1 + 5) * T3), MAXPTS,20000
Set trace 4 equal to the time derivative of the product of T3 and A1 times 30 MV plus 5

T5:DEFINE EQN, HIST(T2),PARAM,sdev
Set trace 5 equal to the histogram of trace 2, with the standard deviation pulse parameter.

T1:DEF? Query the trace equation for trace 1.
T1:DEF EQN, A1", MAXPTS,50000

See below for more sample equations.)

Equation Syntax

The trace equation must exactly follow the syntactic guidelines. The relevent terms and guidelines follow.

source	a channel, trace, or memory number: T1 through T8, M1 through M8, A1, B3, C4, etc. To prevent recursive definitions, only traces with a lower number than that in the prefix may be used as a source. Trace 8 (T8) cannot be used as a source. A source and its constants act as one operand as described below.
multiplicative constant	the coefficient of a source, with or without units. Default is 1 constant having no units. The coefficient may be a decimal number, and may be expressed as an exponential number.

DEFINE (continued) DEF

If the units have operators in them, such as mV/s, they should be enclosed in single quotes. Simple units, such as mS may appear without quotes.

If units are not specified, the multiplicative constant should be followed by the multiplication sign. If the multiplicative constant is 1, but units are used, a constant of 1 must be entered. If both the multiplicative constant and units are trivial, they and the multiplication sign should be left out.

additive constant — a number (with no units) added to a source. The constant may be negative. The default is 0 . If the additive constant is zero, then it and the plus or minus sign should be left out.

Sources used more than once can have different constants for each usage. If a source has nontrivial constants, the multiplicative constant (and its unit, if any) should precede the source, and the additive constant should follow it.

If both constants and the units are trivial, the source should be entered by name alone.

function — one of the following mathematical operators:

+	sum of two waveforms
-	difference of two waveforms
*	product of two waveforms
/	quotient of two waveforms
1/	reciprocal (multiplicative inverse)
ABS	absolute value
AVGC	continuous average **
AVGMAG	average of linear FFT magnitude **
AVGPWS	average of log FFT power spectrum **
AVGPWD	average of log FFT power spectrum density **
AVGS	summed average **
DIFF	differentiation *
ERES	enhanced resolution **
EXP	exponent (base e)
EXP10	exponent (base 10)
EXTREMA	extrema *

DEFINE (continued) DEF

FFTIM	FFT imaginary part **
FFTMAG	linear FFT magnitude **
FFTPHA	FFT phase
FFTPWD	log FFT power spectrum density **
FFTPWS	log FFT power spectrum **
FFTRE	FFT real part **
FFTRI	FFT real and imaginary part **
FLOOR	floor (lower part of an extrema waveform)
HIST	histogram**
INTG	integration *
LIMIT	generates an EXTREMA waveform for use in GO-NO-GO testing
LOG	logarithm (base e)
LOG10	logarithm (base 10)
NEG	negate (additive inverse)
ROOF	roof (upper part) of an extrema waveform
SIGN	sign function (-1, 0, or + 1)
SINX	10-to-1 sin(x)/x interpolation
SMTH1	single-point smoothing (average of bin) *
SMTH3	three-point smoothing *
SMTH5	five-point smoothing *
SMTH7	seven-point smoothing *
SMTH9	nine-point smoothing *
SQR	square
SQRT	square root *
TOFAN	Time of Flight - Analog**
TOFPC	Time of Flight - Pulse count**
TREND	trend**
WINHIST	Window Histogram**

*The remote representations for differentiation, integration, and square root do not use the same non-ASCII characters as the display.

DEFINE (continued) DEF

One equation can use no more than two functions, excluding those associated with constants. When a special function (indicated by **) is used in an equation, only one function is allowed.

operand — that which a function operates upon.
Examples:
in A1+ B2, both A1 and B2 are operands of +
in NEG(T4), T4 is the operand of NEG
in A1 / (B4*T3) , both A1 and (B4*T3) are operands

binary function — a function with two operands: + , -, *, /.

unary function — a function with one operand: all from the above list of functions, except the binary functions. Each unary function is followed immediately by a left parenthesis, its operand,and then a right parenthesis.

special function — functions in the above list marked with **.
The parameter list is specified in separate keywords.
Note that all special functions are unary functions.
Use no more than one special function per equation.

The same functions, sources, constants, and units can be used more than once in an equation. Spaces between these terms and parenthesis are optional.

The following are examples of valid equations (the value of the EQN argument):

A1
T3
1/(A1)
10*A1
10MV*T5
A1*B1
10V*A1-100
NEG(10V/S*A1-100)
(NEG(10V*T1+ 100) + T1)
((A1*B2) /C1)

DEFINE (continued) DEF

```
(( (10*A1+ 10) * (30MV*B1-10) ) / (100 OHM*C1 + 10) )
T7 + SQRT(B3)
SQR(T3) * A1
SQR( T7 * A2 )
```

Notes:

If the trace is being displayed, and data is available, it will be updated immediately. Otherwise, the new equation will be displayed when the trace is next activated.

By reducing the number of points to process, the processing time can be reduced. Starting with the first point of a source, the 7200A uses every n-th point, where n depends on the timebase, the number of points in the source, and the value selected for the maximum number of points.

DEFINE_REPLAY DEFR

Purpose: Defines a trace equation used during Replay Traces, which specifies the source(s) of data and how it is processed.

Command: **prefix: DEFR keyword,value...,keyword,value**

Query: **prefix: DEFR?**

Response: prefix: DEFR keyword,value...,keyword,value

Arguments: prefix The prefix for this command is limited to traces, i.e., T1, T2, ... T8.

keyword Any valid keyword as for the DEFINE command.

Default: The default equations assign the memories to each trace:

T1 = M1 T2 = M2 T3 = M3 T4 = M4
T5 = M5 T6 = M6 T7 = M7 T8 = M8

Examples: **T4:DEFR EQN, DIFF((30MV*M1 + 5) * T3), MAXPTS,20000**
Set trace 4 equal to the time derivative of the product of T3 and M1 times 30 MV plus 5

T5:DEFR EQN, HIST(T2),PARAM,sdev Set trace 5 equal to the histogram of trace 2, with the standard deviation pulse parameter.

T1:DEFR? Query the trace equation for the trace 1.
T1:DEF EQN, M1", MAXPTS,50000

Note: This command's syntax and arguments are identical to the DEFINE command exept that only Memories (M1 thru M8) may be used as sources in the trace equation. Plug-ins and channel numbers are not allowed.

See Also: DEFINE

DELETE_FILE DELF

Purpose: Deletes the specified files on the internal or floppy disk.

Command: **DELETE_FILE keyword,value**

Argument: The argument consists of a keyword followed by its value.

Keyword	Meaning:Value
INPN	Panel settings file on the internal disk filename.ext filename including extension or all files
INAS	Program file on the internal disk filename.ext filename including extension or all files
FLPY	File on the floppy disk filename.ext filename including extension or all files

Examples:

DELF INPN,'MIKE.PNL" Delete the panel settings file MIKE.PNL from the internal disk.

DELF FLPY,"ALL FILES" Delete all files on the floppy disk.

See Also: DIRECTORY_LIST, COPY_FILE

DIRECTORY_LIST DIR

Purpose: Returns a listing of files on the internal or floppy disk.

Query: **DIRECTORY_LIST? argument**

Response: DIRECTORY_LIST "file1.ext","file2.ext",...

Argument: The argument specifies the disk and group of files to be listed:

INPN	Panel settings files on the internal disk
INAS	Program files on the internal disk
FLPY	All files on the floppy disk

Examples: **DIR? FLPY** Returns a list of all files on the floppy disk.
DIR "TRACE1.000", "TRACE1.001", "TRACE.PNL"

DISPLAY DISP

Purpose: Enable/disable display processing of traces and measurements. Disabling display processing may increase acquisition rate and overall throughput while still allowing processing of traces.

Command: **DISPLAY state**

Query: **DISPLAY?**

Response: DISPLAY state

Argument:

state	ON	Causes the traces which are turned on to be updated on the screen.
	OFF	Stops display updates of traces and their associated annotation. Any traces which are on are still processed and can be read out to a host computer or stored to the disk.

Examples:

DISPLAY OFF	Turns off display updates of all traces.
DISPLAY?	Queries whether display processing is enabled or disabled.
DISPLAY ON	Reports that display processing is enabled.

Note: The display is always enabled when the 7200A is powered on or reset.

See Also: TRACE, DEFINE

DISPLAY_UPDATE DISU

Purpose: this command allows you to specify how often the display is updated while a history function (average, histogram, etc.) is collecting data. To increase the throughput, use a longer time between updates.

Command: **DISPLAY_UPDATE time**

Query: **DISPLAY_UPDATE?**

Response: DISPLAY_UPDATE time

Arguments:

time	number of seconds between updates. Range is 1 to 50,000 in increments of 1 second.

Examples:

DISU 5	sets the periodic display update to 5 seconds
DISU?	requests the current display update for history functions.
DISU 1000	Response indicates that the display is updated once every 1000 seconds while a history function is accumulating data.

Note: This applies to all traces.

See Also: DEFINE

DOT_JOIN DTJN

Purpose: Selects whether or not lines should be drawn between acquired data points.

Command: **DOT_JOIN value**

Query: **DOT_JOIN?**

Response: DOT_JOIN value

Arguments:

value	ON	enables the drawing of lines between data points.
	OFF	only the data points will be displayed.

Examples:

DOT_JOIN ON	requests that traces be drawn with the data points connected by lines.
DOT_JOIN?	Queries the method of displaying waveforms.
DOT_JOIN OFF	Indicates only the data points are being drawn.

DPE

Purpose: Sets the Data Processing Enable register (DPE). The DPE register determines which events in the Data Processing Register (DPR) are reported. DPR identifies which internal software processing event(s) has completed. Any reported DPR event sets the DPB summary message bit (bit # 1) of the main Status Byte (STB).

Command: **DPE mask**

Query: **DPE?**

Response: DPE mask

Argument: mask When expressed in binary, this number (between 0 and 1023) represents the bits of the DPR that can be reported:

Bit #	Significance
9	Replay Traces Done
8	Record Traces Done
7	Self-test Done
6	Recall Done
5	Store Done
4	Auto Setup Completed
3	Maximum Sweeps Reached
2	Calibration Completed
1	Waveform Processing Completed
0	Hardcopy Completed

Examples: **DPE 15** Enables the events represented by the lower four bits of this event register (i.e., Hardcopy completed, waveform processing completed, calibration completed, and maximum sweeps reached). Set bit 3, i.e., decimal 8, bit 2 (decimal 4), bit 1 (decimal 2), and bit 0 (decimal 1). Summing the decimal values yields 15.

DPE? Read the contents of the DPE.
DPE 15 Response indicates the contents as 15, the lower four enable bits are set.

See Also: CAE, DPR?, MSE, WPE

DPR?

Purpose: Reads and then clears the contents of the Data Processing event Register (DPR).

The DPR register identifies which internal software processing event(s) has completed.

Some of the DPR register bits are summary bits for other status registers. DPR? will not clear a bit if its corresponding register is not cleared.

Clearing the DPR register also clears the DPB summary message bit (bit # 1) of the main Status Byte (STB).

Query: **DPR?**

Response: DPR value

Argument: value When expressed in binary, this number (between 0 and 1023) represents the bits of the DPR.

The status bytes that each DPR bit summarizes and the bit's assignment follow:

Bit #	Associated Status Byte	Significance
9	none	Replay Traces Done
8	none	Record Traces Done
7	none	Self-test Done
6	none	Recall Done
5	none	Store Done
4	none	Auto Setup Completed
3	MSR	Maximum Sweeps Reached
2	CAR	Calibration Completed
1	WPR	Waveform Processing Completed
0	none	Hardcopy Completed

Bit # 1, Waveform Processing Completed, is a summary message bit that is set as soon as processing for any trace is completed. It is cleared when the WPR register is read.

DPR? (continued)

Bit # 4, Auto Setup Completed, is set when all the plug-ins have been automatically setup.

All registers must be cleared using the *CLS comand before waiting for an event to occur. Otherwise, a previous event may be read.

Examples:

DPR? Read and then clear the DPR register.
DPR 4 Response indicates that calibration is completed.

See Also: ALST?, CAR, DPE, MSR, *STB?, WPR, *CLS

EAE, EBE

Purpose: Each of these commands acts in the same way on its corresponding plug-in event status enable register. The middle letter of the command identifies the corresponding plug-in. Although plug-in A is discussed, the following description applies equally to plug-in B.

The command EAE sets the error mask for plug-in A Enable register. The EAE register determines which events in the Event for plug-in A Register (EAR) are reported. EAR identifies the type of error which occurred in the plug-in. Any reported EAR event sets the EAB summary message bit (bit # 1) of the Device Dependent Register (DDR) and may propagate to the main Status Byte (STB).

Command: **EAE mask, EBE mask**

Query: **EAE?, EBE?**

Response: EAE mask, EBE mask

Argument: mask — When expressed in binary, this number (between 0 and 63) represents the bits of the EAR that can be reported:

Bit #	Significance
5	Calibration Failed
4	External overload
3	Channel 4 overload
2	Channel 3 overload
1	Channel 2 overload
0	Channel 1 overload

Examples:

EAE 15 Enables the events represented by the lower four bits of this event register (i.e., channels 1 through 4 overload). Set bit 3, i.e., decimal 8, bit 2 (decimal 4), bit 1 (decimal 2), and bit 0 (decimal 1). Summing the decimal values yields 15.

EAE? Read the contents of EAE.
EAE 15 Response indicates the contents as 15, the lower four enable bits are set.

See Also: DDR, EAR?, EBR?

EAR?, EBR?

Purpose:

Each of these commands acts in the same way on its corresponding plug-in status register. The middle letter of the command identifies the corresponding plug-in. Although plug-in A is discussed, the following description applies equally to plug-in B

The EAR? query reads and then clears the contents of the Error(s) for plug-in A Register. The EAR specifies the cause of the failure (e.g., channel 1 overload).

Clearing the EAR register also clears the EAB summary message bit (bit # 1) of the Device Dependent Register (DDR) and the effect could propagate to the main Status Byte (STB).

Query:

EAR?, EBR?,

Response:

EAR value, EBR value,

Argument:

value — When expressed in binary, this number (between 0 and 63) represents the bits of the EAR:

Bit #	Significance
5	Calibration Failed
4	External overload
3	Channel 4 overload
2	Channel 3 overload
1	Channel 2 overload
0	Channel 1 overload

Examples:

EBR? Read and then clear the contents of EBR.
EBR 2 Response indicates channel 2 overload.

See Also:

ALST?, DDR, EAE, EBE, *STB

EME

Purpose: Sets the error mask for the Mainframe Enable register (EME). The EME register determines which events in the Event for Mainframe Register (EMR) are reported. EMR identifies the type of error which occurred in the mainframe. Any reported EMR event sets the EMB summary message bit (bit # 1) of the Device Dependent Register (DDR) and propagates to the main Status Byte (STB).

Command: **EME mask**

Query: **EME?**

Response: EME mask

Argument: mask When expressed in binary, this number (between 0 and 63) represents the bits of the EMR that can be reported:

Bit #	Significance
5	System Clock Failed
4	I/O failure
3	Hard disk failure
2	Floppy failure
1	Display failure
0	Internal communication hardware failure

Examples: **EME 15** Enables the events represented by the lower four bits of this event register (i.e., internal communication hardware failure, display failure, floppy failure, and hard disk failure). Set bit 3, i.e., decimal 8, bit 2 (decimal 4), bit 1 (decimal 2), and bit 0 (decimal 1). Summing the decimal values yields 15.

EME? Read the contents of EME.
EME 15 Response indicates the contents as 15, the lower four enable bits are set.

See Also: EMR?

EMR?

Purpose: Reads and then clears the contents of the Error(s) for the Mainframe event Register (EMR). The EMR specifies the cause of the failure (e.g., floppy disk hardware failure).

Clearing the EMR register also clears the EMB summary message bit (bit # 1) of the Device Dependent Register (DDR) and the effect could propagate to the main Status Byte (STB).

Query: **EMR?**

Response: EMR value

Argument: value When expressed in binary, this number (between 0 and 63) represents the bits of the EAR:

Bit #	Significance
5	System Clock Failed
4	I/O failure
3	Hard disk failure
2	Floppy failure
1	Display failure
0	internal communication hardware failure

Examples: **EMR?** Read and then clear the EMR.
EMR 16 Response, i.e., 10 in hexadecimal, indicates that an I/O failure occurred.

See Also: ALST?, DDR, EME, *ESR?, *STB?

EXE

Purpose: Sets the Execution error Enable register (EXE). The EXE register determines which events in the Execution error status Register (EXR) are reported. EXR identifies the type of execution error that has occurred. Any reported EXR event sets the EXB summary message bit (bit # 4) of the standard Event Status Register (ESR) and propagates to the main Status Byte (STB).

Command: **EXE mask**

Query: **EXE?**

Response: EXE mask

Argument: mask When expressed in binary, this number (between 0 and 15) represents the bits of the EXR that can be reported:

Bit #	Significance
3	Fatal Error
2	Internal Error
1	Operator Error
0	Operator Warning

Examples:

EXE 3 Enables the events represented by the lower two bits of this event register (i.e., operator warning, operator error).

EXE? Read the contents of the EXE.
EXE 3 Response to query indicates that the lower two enable bits are set.

See Also: EXR?

EXR?

Purpose: The EXR? query reads and then clears the contents of the Execution error Register (EXR).

EXR identifies the type of execution error that has occurred. Each of four bits in the EXR corresponds to a different category of error/warning conditions. Each category corresponds to a different value queue of error/warning codes. If a queue is reported as having one or more codes present, the corresponding bit in the EXR is set.

Clearing the EXR register also clears the EXB summary message bit (bit # 4) of the main Status Byte (STB).

Query: **EXR?**

Response: EXR value

Argument: value When expressed in binary, this number (between 0 and 15) represents the bits of the EXR:

Bit #	Associated Status Queue	Significance
3	FER	Fatal Error
2	IER	Internal Error
1	OER	Operator Error
0	OWR	Operator Warning

Examples:

EXR? Read and then clear the EXR.
EXR 1 Response indicates that an operator warning error is reported.

See Also: ALST?, *CLS, *ESR?, EXE, *STB?

FER?

Purpose: Reads and then clears the contents of the Fatal Error Register (FER). The FER register identifies the most recent fatal command error. A fatal error occurs when a failure inside the 7200A stops its operation. With some failures, operation can be continued by turning off and on power.

The FER is a queue which contains a unique encoded value. The value corresponds to the fatal error which stopped the 7200A.

Clearing the FER register also clears the FEB summary message bit (bit # 3) of the Execution error Register (EXR) and the effect could propagate to the main Status Byte (STB).

Query: **FER?**

Response: FER value

Argument: value Corresponds to an error. A listing of all encoded values with their meanings is found in Appendix A.

Example: **FER?** Requests the latest fatal error value.
FER 735 Response indicates that the Trace Edit subsystem is inoperable

See Also: ALST?, *CLS, *ESR?, EXE, EXR?, *STB?

FIND_CTR_RANGE FCR

Purpose: Automatically sets the center and width of a histogram or the center and height of a trend to best display the accumulated events.

Command: **prefix:FIND_CTR_RANGE**

Arguments:

prefix	The prefix is limited to traces T1, T2,..., T8

Examples:

T3:FCR	Automatically sets the center and range of the histogram or trend defined in trace 3.

See Also: CENTER_MAX

FORMAT_FLOPPY FFLP

Purpose: Formats the floppy disk in MSDOS compatible format.

Command: **FORMAT_FLOPPY** type

Argument: type specifies the floppy density

HIGH	formats the floppy disk for 1.44 Mbytes
LOW	formats the floppy for 720 kbytes

Notes: formatting a double density disk (DSDD) for HIGH density can result in unreliable operation.

GPIB_ADDRESS GPAD

Purpose: set the 7200A's GPIB address

Command: **GPIB_ADDRESS** addr

Query: **GPIB_ADDRESS?**

Response: GPIB_ADDRESS addr

Argument: addr the GPIB address in the range 0 to 31

Example:

GPIB_ADDRESS 8	Sets the 7200A GPIB address
GPIB_ADDRESS?	Queries the 7200A's GPIB address
GPIB_ADDRESS 6	Reports that the 7200A's GPIB address is 6

Notes: Care should be used when issuing this command from GPIB since it will immediately change the address of the 7200A.

See Also: REM_CTRL

GRID

Purpose:	Selects the number of grids to display. All grids are 8 divisions vertically by 10 divisions horizontally. The number of grids can be independantly specified for the full screen display and for the half screen display (i.e. when waveform parameters are on).	
Command:	**GRID full type, half type**	
Query:	**GRID?**	
Response:	GRID full type, half type	
Argument:	full type	choice of SINGLE, DUAL, QUAD, or OCTAL
	half type	choice of SINGLE, DUAL, QUAD
Example:	**GRID SINGLE,DUAL**	Causes a single grid to be displayed with all traces overlaid when in the full screen display and dual grids for the half screen display
	GRID? GRID QUAD,DUAL	Requests the grid type currently displayed. Indicates that QUAD grid is selected for full screen displays and DUAL grids for half screen displays.
See Also:	CURSOR_SET, VERT_POSITION	

GRID_STYLE GRDS

Purpose: Selects the style of grid to display

Command: **GRID_STYLE value**

Query: **GRID_STYLE?**

Response: GRID_STYLE value

Arguments: choices are: STANDARD, DOTS, CROSSHAIR, and BOX.

Examples:

GRID_STYLE BOX requests that the grid be drawn as a box around the trace area only.

GRID_STYLE? Queries the current grid style.

GRID_STYLE STANDARD

HARDCOPY HCPY

Purpose: Initiates the currently setup hardcopy (screen dump, waveform, or program) to the selected device. If a hardcopy is in progress, it can be aborted by sending this command with the ABORT argument.

Command: **HARDCOPY**

HARDCOPY Abort

Query: **HARDCOPY?**

Response: HARDCOPY status

Argument:

Abort	Terminates immediately any hardcopy in progress.
status	Response to query: 0- no hardcopy in progress, 1- hardcopy in progress Optionally, the Hardcopy Done bit in the Data Processing status register (DPR) may be checked for hardcopy status.

Examples:

HCPY	Initiates a hardcopy to the Port using the HCSU command..
HCPY Abort	Immediately terminates hardcopy.
HCPY? HCPY 0	Requests whether hardcopy is in progress. Response to query indicates that no hardcopy is in progress.

Note: When the Remote Control port is the same as the Hardcopy port, the hardcopy output is sent to the Remote Host. In this case, the 7200A does not enter Talker-Only mode but instead will wait to be addressed to talk by the Remote Host before sending the hardcopy data.

See Also: HARDCOPY_SETUP, INTS

HARDCOPY_SETUP HCSU

Purpose: Configures the 7200A hardcopy driver. The command can specify the device type, transmission mode, plot size, etc. The query has no parameters and returns all possible keyword and value pairs.

Command: **HARDCOPY_SETUP keyword,value,keyword,value,...,keyword ,value**

Query: **HARDCOPY_SETUP?**

Response: HARDCOPY_SETUP keyword,value,keyword,value,...,keyword,value

Arguments: An argument consists of a keyword followed by its value. Any number of arguments in any order may be used to change individual settings.

Keyword	Meaning:Value
PORT	Port used for hardcopy device:
	GPIB See note below.
	RS232 See note below.
	CENT Rear panel Centronics connector.
	FLOPPY Hardcopy to a file on floppy disk. When selected, the following argument indicates the destination file:
	FILE DOS compatible filename:" string."
ANNOT	Annotation ON will place along the top of the grid and waveform display: the date, time, up to a 40 character comment, and the LeCroy logo. Softkey labels are also included. OFF suppresses their inclusion except that softkeys are always printed in setup screens.
COMM	Up to 40 character comment describing the print/plot: string
OPTY	Output type: SCREEN_DUMP, WAVEFORM, PROGRAM

For plotters, use the following:

DEV	Name of the plotter:
	HP7470A Hewlett Packard 7470A or compatible 74xx series
	FP5301 Graphtec FP5301
	PM8151 Philips PM 8151
	HP7550A Hewlett Packard 7550A or compatible 75xx series
SPEED	Drawing rate:
	N normal
	S slow (for slow-drying inks)

HARDCOPY_SETUP (continued) HCSU

PENS Maximum number of pens supported by plotter:
1 through 8

PSIZE Size of the plotter paper needed to fit the plot:
A5 for 5.5" x 8.5"
A4 for 8.5" x 11"
A3 for 11" x 17"
NS or a non-standard size. When selected, the following arguments can be used:
GRID size of grid square when single grid is selected, in mm: 0.1 through 99.9
LLX lower left X-position (mm): -999.9 through 999.9
LLY lower left Y-position (mm): -999.9 through 999.9

For printers, use the following:

DEV Name of the printer:
CIT_120D Citizen 120D
HPQJ Hewlett Packard Quiet Jet
EPSON FX series or compatibles
HP_LASER_JET Hewlett Packard Laser Jet, printer density:
DENS 75_dpi, 100_dpi, 150_dpi, 300_dpi
HP_THINK_JET Hewlett Packard Think Jet, printer density:
DENS SINGLE, DOUBLE

DENS Printer density for other than Laser and Think Jet:
SINGLE, DOUBLE, QUADRUPLE, HIGH_SPEED, CRT, HIGH_RESOLUTION, ONE_TO_ONE, TWO_TO_ONE

PFEED Page feed ON will cause the last page printed to be followed by a form feed. OFF suppresses the final form feed.

PSIZE Size of the printer paper needed to fit the plot:
A5 for 5.5" x 8.5"
NS for a non-standard size. When selected, the following argument can be used:
GRID size of grid square when single grid is selected, in mm: 0.1 through 99.9.

If the output type (OPTY) is WAVEFORM:

TRNO Trace number to print:
T1, T2, T3, T4, T5, T6, T7, T8

HARDCOPY_SETUP (continued) HCSU

WBLK	Part(s) of the waveform to print:	
	ALL	DESC, TEXT, TIME, DAT1, and DAT2 (see below)
	DESC	acquistion settings
	TEXT	user text field in the waveform
	TIME	segmented waveform time arrays
	DAT1	actual waveform data samples
	DAT2	for dual trace waveforms, such as envelopes, prints processed waveform data

Examples:

HCSU PORT,GPIB,SPEED,N,LLX,0,LLY,0

Sets the hardcopy port to GPIB, the speed to normal, and the lower left X- and Y-positions to 0.

HCSU PORT,FLOPPY,FILE,TRACE1,ANNOT,ON,COMM,"Test 1 Data"

Sends the hardcopy to a file called TRACE1.HCP on the floppy disk. The title above the grid is Test 1 Data. Since DEV is not specified, the format of the hardcopy file will correspond to the last selected printer or plotter.

HCSU DEV,HP_LASER_JET,DENS,300_dpi,TRNO,T2,WBLK,DAT1

Uses the Hewlett Packard Laser Jet, with a print density of 300 dpi, to print the waveform data samples of trace 2. Since PORT is not specified, the hardcopy will be sent to the last selected port.

HCSU? Queries all keywords and value pairs.

HCSU PORT,FLOPPY,FILE,TRACE1,DEV,HP7550A,ANNOT,
ON,SPEED,N,PENS,8,PSIZE,NS,GRID,20.0,LLX,150.0,LLY,150.0,COMM,
TEST 1 DATA

HCSU?

HCSU PORT,RS232,DEV,HP7470A,ANNOT,OFF,SPEED,S,PENS,
5,PSIZE,A5,COMM,"7200A DIGITAL OSCILLOSCOPE"

HCSU?

HCSU PORT,CENT,DEV,EPSON,DENS,TWO_TO_ONE,ANNOT,
ON,PFEED,ON,OPTY,SCREEN_DUMP,PSIZE,NS,GRID,15.0,COMM,
7200A DIGITAL OSCILLOSCOPE

HARDCOPY_SETUP (continued) HCSU

HCSU?
HCSU PORT,GPIB,DEV,HP_THINK_JET,PFEED,ON,OPTY,
WAVEFORM,TRNO,T8,WBLK,DESC

HCSU?
HCSU PORT,CENT,DEV,HP_LASER_JET,PFEED,ON,OPTY,
PROGRAM

Note: When the Remote Control port is the same as the Hardcopy port, the hardcopy output is sent to the Remote Host. In this case, the 7200A does not enter Talker-Only mode but instead will wait to be addressed to talk by the Remote Host before sending the hardcopy data

If PORT is set to the Remote Control port, then the remote HCPY command will send the currently selected hardcopy to the Remote Host as a response. This will disable remote control operation from that port until the hardcopy operation is complete. If the Remote Host is GPIB, then the 7200A will wait to be addressed to talk before sending the hardcopy data. However, if the Hardcopy key is pressed, then the 7200A will enter Talker-Only mode and send the data onto the GPIB bus. If PORT is different from the Remote Control port, then the remote HCPY command will work the same as pressing the local Hardcopy key. For example, if the Remote Host port is RS232 and the Hardcopy PORT is GPIB, then sending HCPY from RS232 will put the 7200A into Talker-Only mode and initiate the hardcopy to the GPIB port.

See Also: HARDCOPY, HARDCOPY-TRANS, INTS

HARDCOPY_TRANS HCTR

Purpose: Immediately sends a series of characters to the hardcopy device.

Command: **HARDCOPY_TRANS string**

Argument: string String of characters to send to the hardcopy device.

Example: **HCTR Test.**

Note: The text string must be less than 2048 characters.

See Also: HARDCOPY_SETUP

HIST_ORIENT HISO

Purpose: This command lets you select the display orientation for histograms. In the standard orientation, the histogram is displayed with the zero baseline at one division up from the bottom of the grid and the peaks grow upward. alternately, you can display the baseline near the top of the screen and have the peaks grow downward.

Command: HIST_ORIENT value

Query: HIST_ORIENT?

Response: HIST_ORIENT value

Arguments:

value	STANDARD	draws the zero baseline at one division from the bottom of the grid and peaks grow upward.
	INVERT	draws the zero baseline at one division from top of the grid and the peaks grow downward.

Example:

HISO	STANDARD	Tells the 7200A to display histograms so that the peaks grow upward
HISO?		Requests the current orientation for histograms.
HISO	INVERT	Reports that histograms are displayed with their peaks growing downward.

Note: This affects the display of all histograms. It applies to all traces.

See Also: DEFINE

HOR_MAGNIFY HMAG

Purpose: Sets the horizontal display magnification of the specified trace. Increasing the magnification causes the displayed trace to represent a shorter region of points. A shorter region accentuates the details of that region. Decreasing the Horizontal magnification value reduces its magnification until all its data points are represented on the screen (no magnification). The trace is expanded about the center of the grid.

Command: **prefix:HOR_MAGNIFY mag**

Query: **prefix:HOR_MAGNIFY?**

Response: prefix:HOR_MAGNIFY mag

Argument:

prefix The prefix is limited to traces, i.e., T1, T2, ... T8.

mag Magnification value, specified in a 1,2,5 sequence; that is, 1,2,5,10,20,..., 2000.

Examples:

T1:HMAG 5 Sets the magnification value to 5 which has the same effect on the number of displayed data points as decreasing the time per division by two steps (without changing the sampling rate).

T3:HMAG? Requests the magnification value for trace 3
T3:HMAG 20 Response indicates a value of 20.

Note: For waveforms acquired in sequence mode, a magnification of 2 will represent one segment on the screen. Greater values of magnification will result in displaying shorter regions of that segment. When magnifying waveforms acquired in sequence mode, the displayed waveform represents one segment. The desired segment is selected by HOR_POSITION.

See Also: HOR_POSITION

HOR_POSITION HPOS

Purpose: Sets the horizontal display position (not the trigger delay) of the specified trace. Increasing the horizontal position moves the waveform towards the right of the display while decreasing the value moves it towards the left.

Command: **prefix:HOR_POSITION pos, seg**

Query: **prefix:HOR_POSITION?**

Response: prefix:HOR_POSITION pos, seg

Argument:

prefix — The prefix is limited to traces, i.e., T1, T2, ... T8

pos — Gives the position of the center of the trace relative to the left edge of the grid. Range is 0 (left edge at center) through 1.0 (center of trace is placed on the right edge of the grid) in increments of .01. A setting of 0.5 centers the trace on the screen.

seg — Indicates the segment number to expand for segmented traces. If this argument is not specificed, the current segment is used.

Examples:

T2:HPOS 0.5,3 The center data point of the third segment of the waveform at acquisition is placed at the center of the screen.

T4:HPOS? Requests the position of the center of trace 4.
T4:HPOS 1,5 Response to query is the position of the center data point of acquisition relative to zero position. The center data point of the fifth segment is at the right-most screen position.

See Also: HOR_MAGNIFY

IER?

Purpose: Reads and then clears the contents of the Internal Error Register (IER). The IER register identifies the most recent internal error. An internal error occurs when the 7200A enters an undefined state. Since it is unlikely that this state would ever occur, LeCroy should be immediately informed of the circumstances surrounding the generation of this error.

The IER is a 16-bit queue which contains a unique encoded value. The value corresponds to the most recent internal error.

Clearing the IER register also clears the IEB summary message bit (bit # 2) of the Execution error Register (EXR) and the effect could propagate to the main Status Byte (STB).

Query: **IER?**

Response: IER value

Argument:

value	Corresponds to an error. A listing of all encoded values with their meanings is found in Appendix A.

Example:

IER?	Requests the latest internal error value.
IER 539	Response indicates an internal Pulse Parameter error.

See Also: ALST?, *CLS, *ESR?, EXE, EXR?, *STB?

INE

Purpose: Sets the Internal state change Enable register (INE). The INE register determines which events in the Internal state change Register (INR) are reported.

Each bit in the INR identifies which plug-in received a trigger. Any reported INR event sets the INB summary message bit (bit # 0) of the main Status Byte (STB).

Command: **INE mask**

Query: **INE?**

Response: INE mask

Argument: mask — When expressed in binary, this number (between 0 and 63) represents the bits of the INR that can be reported:

Bit #	Significance
1	Trigger done for plug-in B
0	Trigger done for plug-in A

Examples:

INE 3 — Enables the events represented by the lower two bits of this condition register (i.e., plug-ins A and B received a trigger). Set bit 1 (decimal 2), and bit 0 (decimal 1). Summing the decimal values yields 3.

INE? — Read the contents of the INE
INE 3 — Response to query indicates that a trigger done for plug-ins A and B can be reported.

See Also: INR?

INR?

Purpose: Reads the contents of the Internal state change Register (INR).

Each bit in the INR identifies which plug-in has received a trigger. Since the INR is an event register, any bits stay set until the register is read. After it is read, all the bits are cleared. Once cleared, its summary bit, INB, in the STB is also cleared.

Query: **INR?**

Response: INR value

Argument: value When expressed in binary, this number (between 0 and 63) represents the bits of the INR:

Bit #	Significance
1	Trigger done for plug-in B
0	Trigger done for plug-in A

Examples:

INR? Read the contents of the INR.

INR 1 Response indicates that the trigger is done for plug-in A.

See Also: ALST?, INE, *STB?

INSPECT? INSP?

Purpose: Transfers a waveform or part of a waveform from the 7200A to the host computer in intelligible form. The command is based on the explanation of the format of a waveform given by the template. Each logical data block of a waveform may be inspected by giving its name (e.g. TRIGTIME as mentioned in the template) as an argument to this query.

Query: **prefix: INSPECT?item,item,...,item**

Response: INSPECT item,< ASCII string> ,item,< ASCII string> ,...,item,< ASCII string>

Arguments:

prefix The prefix for this command is limited to traces!i.e., T1, T2, ... T8.

items any logical data block in the TEMPLATE, for example

ALL	all the waveform blocks
DESC	waveform descriptor(acquisition settings)
TEXT	user defined trace annotation
TIME	RIS time descriptor
DAT1, DAT2	1st and 2nd blocks of waveform data

or any field within the descriptor
If no argument is specified, the 7200A responds as if ALL is sent.

Examples:

T1:INSP? TIMEBASE	Query the timebase setting for Trace1
T1:INSP TIMEBASE:500_us/div	Response indicates Timbase set to 500 µs/div
T1:INSP? DESC	Query the entire waveform Descriptor Response is a formatted string over 300 bytes.

INSPECT? (continued) INSP?

T1:INSP? TRIGGER_COUPLING,TRIGGER_SOURCE,TRIGGER_SLOPE

T1:INSP TRIGGER_COUPLING:AC,TRIGGER_SOURCE:CHANNEL_1,
TRIGGER_SLOPE:POSITIVE

Notes:

The entire response is an ASCII string.

Acts in a manner similar to the WAVEFORM? except INSP? returns text that can be read by the user.

Use the query TEMPLATE? to obtain an up-to-date copy of arguments.

This most basic form of waveform readout supports even the sophisticated user who wishes to verify his understanding and interpretation of the template and descriptor. Refer to Section 3: Waveform Transfer for details.

See Also:

WAVEFORM?,TEMPLATE?, TRACE_ANOT

INTENSITY INTS

Purpose: Sets the intensity level of (a) the grid, (b) the trace and the text, or (c) the background. The intensity level is expressed as a percentage. A level of 100 corresponds to the maximum intensity while a level of 0 sets the intensity to its minimum.

Command: **INTENSITY keyword,value,keyword,value, keyword, value**

Query: **INTENSITY?**

Response: INTENSITY keyword,value,keyword,value, keyword, value

Arguments:

keyword	May be either TRACE, GRID, or BKG
value	May be from 0 (minimum intensity) through 100 (maximum intensity).

Examples:

INTS GRID,65	Sets the intensity of the grid to 65%.
INTS?	Requests the intensities of both grid and trace.
INTS GRID,65,TRACE,90,BKG,30	Reports that the grid is at 65% intensity, the trace is at 90%, and the background is at 30%.

Notes: An argument has two parts: keyword followed by value. Any number of keyword, value pairs may be used in any order to change individual settings. The 7200A returns all values.

When doing a hardcopy, setting the grid intensity to 0 causes the plot to be done without a grid.

See Also: HARDCOPY

LOCAL LOC

Purpose: Places the 7200A into Local mode. In Local mode all front panel controls are operational and the 7200A will accept and execute all remote commands. However, the operator is cautioned against sending remote commands and pressing local keys at the same time since these events occur asynchronously and may produce unexpected results.

Command; **LOCAL**

Examples: **LOC** Puts the 7200A into Local mode

Notes: The 7200A may also be put into Local mode at any time if REN (Remote Enable) is deasserted.

If the GPIB interface is currently in the Remote state, sending the GTL interface message while addressing the 7200A to Listen will put the 7200A into Local mode.

If the GPIB interface is currently in the Remote with Local Lockout state, sending the GTL interface message will place the 7200A into Local with Local Lockout mode. Addressing the 7200A to Listen while in Local with Local Lockout puts the GPIB interface into Remote with Local Lockout.

Sending LOCAL will always put the 7200A into Local mode. If LOCAL is sent with REN asserted, the GPIB interface is put into Remote mode. If LOCAL is sent with Ren deasserted, the GPIB interface is put into Local mode.

See IEEE-488.1(section 2.8) for more details concerning GPIB state transitions.

See Also: REMOTE, LOCKOUT

LOCKOUT LLOK

Purpose: Places the 7200A into Remote with Local Lockout mode. In this mode all local keys on the mainframe and plug-in(s) are disabled and only remote commands are accepted. Once Remote with Local Lockout is set, it can only be cleared when the 7200A is put into Local mode or Remote mode without Local Lockout. See the Local and Remote commands which explain how this is done.

Command: **LOCKOUT**

Examples: **LLOK** Puts the 7200A into Remote with Local Lockout

Notes: The 7200A also switches to Remote with Local Lockout if we are in Local mode and the computer addresses it to Listen and sends the LLO GPIB interface message. However, this will only happen if the GPIB interface bus had been set to the local state when the LLO message was sent.

When the GPIB interface is set to Remote mode it cannot be set to Remote with local Lockout mode, even though the LOCKOUT will place the 7200A into Remote with Local Lockout mode. The GPIB interface can only be put into Remote with Local Lockout from the Local state. When the GPIB interface is put into Remote with Local Lockout, it forces the 7200A into Remote with Local Lockout.

When the GPIB interface is in Remote with Local Lockout, deasserting REN will put the GPIB interface into Local mode. Sending the GTL interface message puts the GPIB interface into the Local with Local Lockout mode.

Sending LOCKOUT will always put the 7200A into Remote with Local Lockout mode. If LOCKOUT is sent with REN asserted, the GPIB interface is put into Remote mode. If LOCKOUT is sent with REN deasserted, the GPIB interface is put into Local mode.

See IEEE-488.1 (Section 2.8) for more details concerning GPIB state transitions.

See Also: REMOTE, LOCAL

MSE

Purpose: Sets the Maximum Sweeps Event enable register (MSE). The MSE register determines which events in the Maximum Sweeps Register (MSR) are reported. The MSR identifies which trace has reached its maximum number of sweeps. Any reported MSR event sets the MSB summary message bit (bit # 3) of the Data Processing Register (DPR) and propagates to the main Status Byte (STB).

Command: **MSE mask**

Query: **MSE?**

Response: MSE mask

Argument: mask When expressed in binary, this number (between 0 and 255) represents the bits of the Maximum Sweeps Register the user wants reported to the DPR:

Bit #	Significance
7	Maximum sweeps reached for Trace 8
6	Maximum sweeps reached for Trace 7
5	Maximum sweeps reached for Trace 6
4	Maximum sweeps reached for Trace 5
3	Maximum sweeps reached for Trace 4
2	Maximum sweeps reached for Trace 3
1	Maximum sweeps reached for Trace 2
0	Maximum sweeps reached for Trace 1

Examples: **MSE 15** Enables the events represented by the lower four bits of this event register (i.e., maximum sweeps reached for traces 1 through 4). Set bit 3, i.e., decimal 8, bit 2 (decimal 4),bit 1 (decimal 2), and bit 0 (decimal 1). Summing the decimal values yields 15.

MSE? Read the contents of the MSE.
MSE 15 Response to query indicates the lower four enable bits are set.

See Also: MSR?, WPR?, WPE

MSR?

Purpose: Reads and then clears the Maximum Sweeps Register (MSR) which identifies the trace for which the maximum sweeps has been reached.

This applies to summation averaging, histograms, envelope, and any other routines which require a history to be accumulated.

Clearing the MSR register also clears the MSB summary message bit (bit # 3) of the Data Processing Register (DPR) and the effect could propagate to the main Status Byte (STB).

Query: **MSR?**

Response: MSR value

Argument: value When expressed in binary, this number (between 0 and 255) represents the bits of the MSR:

Bit #	Significance
7	Maximum sweeps reached for Trace 8
6	Maximum sweeps reached for Trace 7
5	Maximum sweeps reached for Trace 6
4	Maximum sweeps reached for Trace 5
3	Maximum sweeps reached for Trace 4
2	Maximum sweeps reached for Trace 3
1	Maximum sweeps reached for Trace 2
0	Maximum sweeps reached for Trace 1

Examples:

MSR? Read the contents of the MSR.

MSR 3 Response indicates that the maximum sweeps are reached for traces 1 and 2.

See Also: ALST?, DPE, DPR?, MSE, *STB?

MULTI_ZOOM MZOM

Purpose: Allows the selected traces to be simultaneously expanded and repositioned, horizontally and/or vertically whenever a command from either the front panel or remote host is issued for any trace in the group. The group is defined by all traces which are enabled by MULTI_ZOOM_SETUP.

Command: **MULTI_ZOOM value**

Query: **MULTI_ZOOM?**

Response: MZOM value

Argument:

value	ON	Turns multi zoom on
	OFF	Turns multi zoom off

Examples:

MZOM ON	Turns on multi zoom
MZOM?	Requests multi zoom status
MZOM off	indicates multi zoom is off

See Also: MULTI_ZOOM_SETUP, HOR_MAGNIFY, HOR_POSITION, VERT_POSITION, VERT_MAGNIFY

MULTI_ZOOM_SETUP MZSU

Purpose: Groups traces together for simultaneous expansion and repositioning. Allows the user to select which axis, (horizontal, vertical or both horizontal and vertical) the trace group will be affected when MULTI_ZOOM is enabled.

Selects which traces are in the group which will be simultaneously expanded and repositioned, horizontally and/or vertically when multi zoom is on.

Command: **MULTI_ZOOM_SETUP keyword, value, keyword, value,..., keyword, value**

Query: **MULTI_ZOOM_SETUP?**

Response: MULTI_ZOOM_SETUP keyword, value, keyword, value,..., keyword, value

Argument:

Keyword	meaning	values
LOCK	selects horizontal, vertical, or both	HOR, VERT HOR_VERT
T1, T2,..., T8	trace number to add/delete	ON or OFF

Examples:

MZSU LOCK, HOR, T1, ON, T2, ON traces 1 and 2 locks together for horizontal expansion only

MZSU? requests lock and trace status

MZSU LOCK, VERT, T1, OFF, T2, ON, T3, ON, T4, ON,
T5, OFF, T6, OFF, T7, OFF, T8, OFF

indicates traces 2,3, and 4 are locked for Vertical expansion

See Also: MULTI_ZOOM, HOR_MAGNIFY. HOR_POSITION, VERT_POSITION, VERT_MAGNIFY

OER?

Purpose: Reads and then clears the contents of the Operator Error Register (OER). The OER register identifies the most recent operator error. An operator error occurs when a command cannot be executed because it contains an illegal request. For example, an argument is outside its required range.

The OER is a queue which contains a unique encoded value. Each value corresponds to a particular operator error.

Clearing the OER register also clears the OEB summary message bit (bit # 1) of the Execution error Register (EXR) and the effect could propagate to the main Status Byte (STB).

Query: **OER?**

Response: OER value

Argument:

value	Corresponds to an error. A listing of all encoded values with their meanings is found in Appendix A.

Example:

OER?	Requests the latest operator error value.
OER 200	Response indicates an error parsing the remote command.

See Also: ALST?, *CLS, *ESR?, EXE, EXR?, *STB?

OWR?

Purpose: Reads and then clears the contents of the Operator Warning Register (OWR). The OWR register identifies the most recent operator warning error. An operator warning error occurs when a command contains an illegal request but is automatically corrected. The command is completed using the correction. For example, if an argument is sent that is between its defined steps, the 7200A will complete the command using the step closest to the requested value.

The OWR is a queue which contains a unique encoded value. The value corresponds to the most recent operator warning.

Clearing the OWR register also clears the OWB summary message bit (bit # 0) of the Execution error Register (EXR) and the effect could propagate to the main Status Byte (STB).

Query: **OWR?**

Response: OWR value

Argument:

value	Corresponds to an operator warning. A listing of all encoded values with their meanings is found in Appendix A.

Example:

OWR?	Requests the latest operator warning value.
OWR 2	Response indicates that a command was sent which attempted to set a value outside its bounds. The value was adjusted to its limit and used as such in subsequent operations.

See Also: ALST?, *CLS, *ESR?, EXE, EXR?, *STB?

PARAMETER_ADD PAAD

Purpose: Adds a waveform parameter to the list of parameters calculated when the displayed cursor type (as set by CURSOR_MEASURE) is Extended Parameters. These parameters are recalculated each time the trace is updated. The calculation is based on a section of the trace that is delineated by two cursors (as set by CURSOR_SET).

Command: **prefix:PARAMETER_ADD parameter [,keyword,value] [,keyword,value] [,position], ..., parameter [,keyword,value] [,keyword,value] [,position]**

Query: **prefix:PARAMETER_ADD?**

Response: prefix:PARAMETER_ADD parameter [,keyword,value] [,keyword,value] ,position, ..., parameter [,keyword,value] [,keyword,value] ,position

Arguments:

prefix — The prefix is limited to traces, i.e., T1, T2, ... T8.

parameter — The available parameters are listed below.

keyword — Some parameters have arguments which may be specified bY keyword, value pairs following the parameter. These are described below. If an argument is not specified, the default value is used.

position — The position in which the parameter will appear in the Extended Parameter display may optionally be specified as P1, P2, ... P20, where the positions are located as follows:

P1	P11
P2	P12
P3	P13
P4	P14
etc.	

If the position is not specified, the next available location will be used.

PARAMETER_ADD (continued) PAAD

General Parameters

parameter	definition + arguments
DATA	The data value at the left cursor. When this parameter is used for histograming, and EVENTS PER WAVEFORM is set to ALL, all of the data values between the cursors are histogrammed.
DATE	Date of acquisition of the waveform.
DUR	For a single sweep waveform, dur is 0. For a sequence waveform, dur is the time from the trigger of the first segment to the trigger of the last segment. For a single segment of a sequence waveform, dur is the time from the trigger of the previous segment to the trigger of the current segmewnt. For a waveform produced by a history function (e.g. AVGS), dur is the time from the trigger of the first waveform accumulated to the trigger of the last waveform accumulated.
FRST	Horizontal position of first (leftmost) cursor.
LAST	Horizontal position of last (rightmost) cursor.
PNTS	Number of points between the cursors.
TIME	Time of acquisition of the waveform.

Time Domain Parameters

keyword	definition + arguments
AMPL	Top minus the base.
AREA	Sum of sampled values between the cursors times the duration of a sample.
BASE	Lower of two most probable states. This is characteristic of rectangular waveforms and represents the lower most probable state determined from the statistical distribution of data point values in the waveform.

PARAMETER_ADD (continued) PAAD

CYCL Number of pairs of transitions in the same direction.

DLY Time from trigger to the midpoint of the first transition.

DNL Differential nonlinearity. Finds the maximum absolute difference between adjacent measured code levels and the ideal code level. The input is assumed to be a Digital to Analog Converter output. The results are measured in terms of LSB's.

keyword	meaning
STEPS	the number of ideal quantization levels. Allowable values are 1 to 256
DWELL TIME	the amount of time that should be spent at each plateau. Allowable values are 0 to 1e6 µs

DUTY Average duration above midpoint value as a percentage of period.

FALL Duration of the pulse waveform's falling transition between two user-specified thresholds, averaged for all falling transitions between the cursors. The thresholds are specified as a percentage of the amplitude.

keyword	meaning
LOW	lower threshold percentage of amplitude 1 - 45 % default = 10%
HIGH	upper threshold percentage of amplitude 55 - 99 % default = 90%

FREQ Reciprocal of period.

PARAMETER_ADD (continued) PAAD

INL — Integral nonlinearity. Finds the maximum absolute difference between each of the measured code levels and its ideal value. The input is assumed to be a Digital to Analog Converter Output. The results are measured in terms of percentage.

keyword	meaning
STEPS	the number of ideal quantization levels. Allowable values are 1 to 256
DWELL TIME	the amount of time that should be spent at each plateau. Allowable values are 0 to 1e6 µs

LMAX — Local maximum. Average of all local maxima between the cursors.

keyword	meaning
HYS	hysteresis controls feature finding discriminates features from noise .01 - 8.0 div default = 0.5div

LMIN — Local minimum. Average of all local minima between the cursors.

keyword	meaning
HYS	hysteresis controls feature finding discriminates features from noise .01 - 8.0 div default = 0.5div

LPP — Local peak to peak (Imax-Imin). Average for all pairs of local maxima and minima between the cursors.

PARAMETER_ADD (continued) PAAD

keyword	meaning
HYS	hysteresis controls feature finding discriminates features from noise .01 - 8.0 div default = 0.5 div

LTBP Local time between peaks. Average for all features between the cursors.

keyword	meaning
HYS	hysteresis controls feature finding discriminates features from noise .01 - 8.0 div default = 0.5 div

LTBT Local time between troughs. Average for all features between the cursors.

keyword	meaning
HYS	hysteresis controls feature finding discriminates features from noise .01 - 8.0 div default = 0.5 div

LTMN Local time at minimum. Time from trigger to first local minimum between the cursors.

keyword	meaning
HYS	hysteresis controls feature finding discriminates features from noise .01 - 8.0 div default = 0.5 div

LTMX Local time at maximum. Time from trigger to first local maximum between the cursors.

keyword	meaning
HYS	hysteresis controls feature finding discriminates features from noise .01 - 8.0 div default = 0.5 div

PARAMETER_ADD (continued) PAAD

LTOT — Local time over threshold. Time over a user-specified percentage of the local peak to peak range, averaged for all features between the cursors.

keyword	meaning
HYS	hysteresis controls feature finding discriminates features from noise .01 - 8.0 div default = 0.5 div
THR	threshold percentage of peak to peak 0-100 % default = 50%

LTUT — Local time under threshold. Time under a user-specified percentage of the local peak to peak range, averaged for all features between the cursors.

keyword	meaning
HYS	hysteresis controls feature finding discriminates features from noise .01 - 8.0 div default = 0.5 div
THR	threshold percentage of peak to peak 0-100 % default = 50%

MAX — Maximum value of the waveform between the cursors.

MEAN — Average or DC level of the waveform. If the waveform is periodic, it is computed over an integral number of periods.

MEDI — The median value is computed over an integral number of periods if the waveform is periodic.

MIN — Minimum value of thewaveform between the cursors.

MODE — The mode is computed over an integral number of periods if the waveform is periodic

PARAMETER_ADD (continued) PAAD

NBPH — Narrow-band phase in degrees relative to the left cursor for a user-specified frequency.

keyword	meaning
FRQ	Center Frequency .001 - 1000000kHz default 1kHz

NBPW — Narrow-band power in dbV (relative to 1V rms) for a user-specified frequency. It is computed over an integral number of periods.

keyword	meaning
FRQ	Center Frequency .001 - 1000000kHz default 1kHz

OVSN — Overshoot negative- Base value minus the minimum sample value, as a percentage of the amplitude.

OVSP — Overshoot positive- Maximum sample value minus the top value, as a percentage of the amplitude.

PER — Time of a full cycle averaged for all full cycles between the cursors.

PKPK — Difference between the maximum and minimum values.

RISE — Duration of the pulse waveform's rising transition between two user-specified thresholds, averaged for all rising transitions between the cursors. The thresholds are specified as a percentage of the amplitude.

keyword	meaning
LOW	lower threshold percentage of amplitude

PARAMETER_ADD (continued) PAAD

	HIGH	1 - 45 % default = 10% upper threshold percentage of amplitude 55 - 99 % default = 90%
RMS	Square-root of sum of squares divided by number of points. If the waveform is periodic, it is computed over an integral number of periods.	
SDEV	Square-root of sum of squares of difference from mean, divided by number of points-1. If the waveform is periodic, it is computed over an integral number of periods.	
TAFL	Time at fall time relative to trigger of the midpoint of the first falling transition	
TARS	Time at rise relative to trigger of the midpoint of the first rising transition	
XAMN	Time relative to trigger where the minimum sample occurred.	
XAMX	Time relative to trigger where the maximum sample occurred.	
TOP	Upper of two most probable states. This is characteristic of rectangular waveforms and represents the higher most probable state determined from the statistical distribution of data point values in the waveform.	
WID	Width of the first pulse (either positive or negative), averaged for similar pulses between the cursors.	

Frequency Domain Parameters

parameter	definition + arguments
MAX	Maximum value of the waveform between the cursors (i.e. amplitude of the largest frequency component).
TPWR	Total power. Area under the power density spectrum. This parameter only applies to spectra produced by the processing function FFTPWD.

PARAMETER_ADD (continued) PAAD

XAMN	Frequency at minumum amplitude.
XAMX	Frequency at maximum amplitude

Histogram Parameters

parameter	definition + arguments
AMPL	Amplitude. Top minus base.
BASE	The centroid of the leftmost significant peak.
FWHM	Full width at half max. The width of the distribution surrounding the mode including values which are at least 1/2 of the maximum value.
FWXX	Full width at a user-specified percentage of the maximum value.

keyword	meaning
THR	threshold percentage of maximum value 0 -100% default = 50%

MAX	Horizontal coordinate of rightmost non-zero bin.
MAXP	Maximum population in any histogram bin (i.e.vertical value at mode).
MEAN	Horizontal centroid of the distribution.
MEDI	Horizontal median. Horizontal value of the midpoint of the distribution.
MIN	Horizontal coordinate of leftmost non-zero bin.
MODE	Horizontal coordinate of bin with maximum population.
PCTL	Percentile. The horizontal coordinate to the left of which lies a user-specified percentage of the distribution lies to the left.

PARAMETER_ADD (continued) PAAD

keyword	meaning
THR	threshold percentage of distribution 0 - 99 % default = 95%

PKPK Peak-to-peak. Horizontal difference between the maximum and minimum values.

PKS The number of peaks in the distribution.

RMS Histogram root mean square. Square-root of sum of squares divided by number of values, computed on the distribution.

SDEV Histogram standard deviation. Square-root of sum of squares of difference from mean, divided by number of values - 1, computed on the distribution.

TOP The Centroid of the rightmost significant peak

TOTP Total population in the histogram.

XAPK The horizontal mean of the user-specified peak. The peak is specified by number, where the peaks are sorted by decreasing area above the background.
(i.e. the peak with the largest area is number 1,
the next largest peak is number 2, etc.)

keyword	meaning
PEAK	number of the peak 1 - 100 default = 1

Examples:

T1:PAAD MAX,P1 Adds the parameter MAX to be computed on trace 1 and displayed in the upper left position.

T1:PAAD RISE,LOW,20,HIGH,80,P11
Adds the parameter RISE to compute the 20% - 80% rise time of trace 1. The parameter will be displayed in the upper right position.

T1:PAAD? Queries the list of parameters for trace 1.

PARAMETER_ADD (continued) PAAD

T1:PAAD MAX,P1,RISE,LOW,20,HIGH,80,P11
Response indicates the identity and position of all the parameters calculated for trace 1 when Extended Parameters are enabled

See Also: CURSOR_MEASURE, CURSOR_SET, PARAMETER_DEL, PARAMETER_CLR, PARAMETER_VALUE

PARAMETER_AVG PAAV

Purpose: Turns waveform parameter averaging on or off.

Command: **PARAMETER_AVG argument**

Query: **PARAMETER_AVG?**

Response: PARAMETER_AVG argument

Argument: The argument specifies whether to turn averaging on or off:
ON
OFF

Examples: **PAAV ON** Turns on parameter averaging.

PARAMETER_CLR PACL

Purpose: Deletes all parameters for all traces in the Extended Parameter display.

Command: **PARAMETER_CLR**

See Also: PARAMETER_ADD,PARAMETER_DEL

PARAMETER_DEL PADL

Purpose: Deletes a parameter from the list of parameters calculated for the specified trace when the displayed cursor type (as set by CURSOR_MEASURE) is Extended Parameters. If no parameter is specified in the command, then all parameters for the specified trace are deleted.

Command: **prefix:PARAMETER_DEL[parameter,[keyword,value,][keyword, value,]..., parameter, [keyword,value,] [keyword,value,]]**

Arguments:

prefix	The prefix is limited to traces T1, T2, ... T8.
parameter	The available parameters are listed in the description of the PARAMETER_ADD command on p.5-86
keyword	Some parameters have arguments which may be specified by keyword,value pairs following the parameter. These are described in the list of parameters. If an argument is specified, only those occurrences of the parameter with the same argument will be deleted. If an argument is not specified, all occurrences of the parameter will be deleted.

Examples:

T1:PADL MAX	Deletes the parameter MAX from the list of parameters to be computed on trace 1.
T1:PADL PCTL,THR,90	Deletes the parameter PCTL (percentile) with 90% threshold from the list of parameters for trace 1. Other PCTL parameters with different thresholds will not be deleted.
T1:PADL PCTL	Deletes all occurrences of PCTL with any threshold from the list of parameters for trace 1.
T1:PADL	Deletes all parameters for trace 1.

See Also: PARAMETER_CLR, PARAMETER_ADD

PARAMETER_VALUE? PAVA?

Purpose: Returns the current value(s) of the waveform parameter(s) for the specified trace. Any parameter on any trace may be queried, whether or not the 7200A is currently computing that parameter. If no parameter is specified in the query, then the values of all the currently specified Extended Parameters for the specified trace are returned. The parameter is calculated on the screen of the trace that is delineated by the Extended Parameter cursors (as set by CURSOR_SET).

Query: **prefix:PARAMETER_VALUE [parameter,[keyword,value,][keyword, value,]..., parameter,[keyword,value,] [keyword,value,]]**

Response: prefix:PARAMETER_VALUE parameter,[keyword,value,][keyword,value,] parameter-value,state ... parameter,[keyword,value,][keyword,value,] parameter-value,state

Arguments:

prefix	The prefix is limited to traces T1, T2, ... T8.
parameter	The available parameters are listed in the description of the PARAMETER_ADD command on p.5-86.
keyword	Some parameters have arguments which may be specified by keyword,value pairs following the parameter. These are described in the list of parameters. If an argument is not specified, the default value is used.
parameter-value	The decimal value and units of the parameter

PARAMETER_VALUE? (continued) PAVA?

state — Indicates additional information about the parameter value as follows:

state	parameter computation state
OK	determined without a problem
AV	averaged over several (up to 500 periods)
PT	computed over integral number of periods
IV	invalid value (insufficient data provided)
NP	not a pulse waveform
LT	actual value is less than given value
OF	signal partially in overflow
UF	signal partially in underflow
OU	signal partially in overflow and underflow
GT	actual value is greater than given value
UD	signal contains undefined points

Examples:

T1:PAVA? MAX — Requests the value of the parameter MAX on trace 1.

T1:PAVA MAX,4.3mV,OK — Response indicates the MAX value is 4.3mV; value was determined without a problem.

T2:PAVA? RISE,LOW,15,HIGH,85
Requests the 15% - 85% rise time of trace 2.

T2:PAVA RISE,LOW,15,HIGH,85,3.6NS,AV
Response indicates that the 15% - 85% rise time of trace 2 is 3.6ns; this value was determined by averaging all the rise times between the cursors.

Note: The parameters are calculated on the section of the trace that is delineated by the Extended Parameter cursors (as set by the CURSOR_SET command). If the trace is a sequence waveform, the parameters are calculated on the first segment between the cursors (i.e., the segment in which the left-most cursor is located).

See Also: PARAMETER_ADD, PARAMETER_DEL, PARAMETER_CLR, CURSOR_SET

PER_CURSOR_SET PECS

Purpose: Positions any one of the six independent cursors at a given screen location in persistence mode. Cursor positions are specified relative to a grid. The positions of the cursors can be modified or queried even if the required cursor is not currently displayed on the screen, or persistence mode is off.

Command: **prefix:PER_CURSOR_SET keyword, position,...,keyword,position**

Query: **prefix:PER_CURSOR_SET?keyword,keyword,keyword,...**

Response: prefix:PER_CURSOR_SET keyword,position,...,keyword,position

Arguments: prefix The prefix is limited to traces, i.e., T1, T2, ..., T8

keyword,position,..., keyword, position

Cursor Type	keyword	position
Vertical Absolute	VABS	-4 to 4 DIV
Vertical Relative	VREF, VDIF	-4 to 4 DIV
Horizontal Absolute	HABS	0 to 10 DIV
Horizontal Relative	HREF, HDIF	0 to 10 DIV

The four cursor types measure the following:

Vertical Absolute	measures the absolute vertical value at a given point.
Vertical Relative	measures the difference between vertical positions of the cursors of a trace(s).
Horizontal Absolute	measures the absolute horizontal position at a given point on the trace(s).
Horizontal Relative	measures the difference between the horizontal positions of two Horizontal cursors.

PER_CURSOR_SET (continued) PECS

Examples:

T1:PECS HABS, 5	Places the horizontal absolute cursor in the center of the waveform.
T1:PECS? HREF	Requests the position of the horizontal reference cursor on trace 1.
T1:PECS HREF,2	The response indicates the position is two divisions to the right of the grid's left edge.

Note:

Keywords ending in REF refer to a reference cursor which is a line of alternating dots and dashes. Keywords ending in DIF correspond to the difference cursor which contains dashes.

A command argument has two parts: Keyword followed by position. Any number of arguments may be used in any order to change or query individual settings.

With query, if no keyword is specified, the positions of all cursors are returned.

When in XY mode with persistence, use the XY cursors to make measurements.

See Also:

CURSOR_MEASURE, PER_CURSOR_VALUE, PERSIST, XY_CURSOR_SET

PER_CURSOR_VALUE? PECV?

Purpose: Returns the values of the specified cursor measurement(s) for a given trace on the persistence display. The corresponding units accompany each reported value.

Query: **prefix: PER_CURSOR_VALUE? keyword,keyword,...,keyword**

Response: prefix:PER_CURSOR_VALUE keyword,value,value,keyword,value,...

Arguments: prefix The prefix is limited to traces, i.e., T1,T2,... T8.

keyword,...,keyword

keyword		meaning
VABS	Vertical Absolute	measures the absolute vertical value at a given point
VREL	Vertical Relative	measures the difference between the vertical positions of two cursors
HABS	Horizontal Absolute	measures the absolute horizontal position of a point on a trace.
HREL	Horizontal Relative	measures the difference between the horizontal positions of two horizontal cursors
value	indicates the cursor measurement results with the correct units.	

Examples:

T1:PECV? HREL Requests the measurement for Horizontal relative cursor positions.

T1:PECV HREL,21.3 MS indicates the difference between the co-ordinates of the two Horizontal cursors .

T1:PECV? VABS Requests the measurement for the Vertical absolute cursor position.

T1:PECV VABS, -24 mV

Notes: If no keywords are specified, the values of all measurements are returned.

PER_CURSOR_VALUE? (continued) PECV?

Any number of cursor types can be specified in the argument. Their cursor names followed by the value(s) and units are returned in the same order as requested.

To measure a cursor value, its cursor type need not be selected by the CURSOR_MEASURE command.

If the cursor is not on the specified trace or the trace is not valid, the value UNDEF is returned.

When in XY mode with persistence, use the XY cursors to make measurements.

See Also: CURSOR_MEASURE, PER_CURSOR_SET, PERSIST, XY_CURSOR_VALUE

PERSIST PERS

Purpose: Enables or disables the persistence display mode.

Command: **PERSIST state**

Query: **PERSIST?**

Response: PERSIST state

Argument:

state	ON	Enables persistence display.
	OFF	Disables persistence display.

Examples:

PERS ON Enables persistence display mode.

PERS? Queries whether persistence is enabled or disabled.

PERS ON Reports that persistence display is enabled.

Note: If persistence is turned on when XY mode is on, successive sweeps of the X vs Y plot will be overlaid.

See Also: PERSIST_SETUP, XY_DISPLAY

PERSIST_SETUP PESU

Purpose: Sets the number of sweeps used in the persistence display and XY persistence.

Command: **PERSIST_SETUP SWEEPS,value**

Query: **PERSIST_SETUP?**

Response: PERSIST_SETUP SWEEPS,value

Argument: An argument consists of a keyword followed by it's value.

Keyword	Meanining:Value
SWEEPS	Number of sweeps to be retained in the persistence display. Valid values are 2 - 200 (in 1-2-5 sequence), INFINITE or AUTO_STOP.

Examples:

PESU SWEEPS,5	Sets persistence display to retain the last 5 sweeps.
PESU?	Queries current persistence setup.
PESU SWEEPS,10	Reports that persistence display mode will retain the last10 sweeps.

See Also: PERSIST, XY

PROG_ARG PRAR

Purpose: Set or retrieve a string. The string is intended to be used to communicate data from one ICL program to another.

Command: **PROG_ARG string**

Query: **PROG_ARG?**

Arguments: string: A quoted string of less than 80 characters.

Examples:

PROG_ARG Any text.	Stores the string Any text. in a fixed location.
PRAR?	Returns the previously stored string Any text.

Notes: An ICL program may recall another ICL program using the PROG_SETUP and PROG_RECALL remote commands. The new program is started as soon as it is loaded; this is the way programs can be chained together. The PROG_ARG command and its query can be used to communicate a small amount of information between chained programs.

See Also: PROG_SET_UP, PROG_RECALL

PROG_CLEAR PRCL

Purpose: Erases the current program. If the program setup screen is displayed, the program will be erased from the screen. This command has no effect on any copy of the program which may have been stored on disk.

Command: **PROG_CLEAR**

Example: **PRCL** Clears the current program.

PROG_COMPILE PRCO

Purpose: compile a program. The query form compiles a program and returns a result string indicating erros, if any.

Command: **PROG COMPILE**

Query: **PROG COMPILE?**

Response: PROG COMPILE result

Examples:

PRCO Compiles a program as specified by the current value of hte PROG SETUP parameters DISK and IFILE.

PRCO? Requexts the currently specified program to be compiled and returns a string indicating whether any erros were found.

PRCO OK Indicates that the program compiled successfully.

PRCO? Compile the currently specified program.

PRCO "Expected variable at line 6"

Indicating there is an error in the program. The error message returned as a string, reports that a variable is missing is in line 6.

See Also: **PROG_SETUP**

PROG_LIST PRLI

Purpose:	Report the list of program files on the currently selected disk.	
Query:	**PROG_LIST?**	
Example:	**PRLI?**	Requests the list of possible input files on the currently selected disk.
	PRLI BABYSIT DEMO	Reports that programs BABYSIT and DEMO are available.
See Also:	DIRECTORY_LIST	

PROG_MODE PRMO

Purpose: Sets the operating mode for program. This command can be used to begin the learn process or start a program executing.

If the program setup screen is displayed when this command is used to change the mode to LEARN or RUN, the main screen will be displayed.

If a program currently exists when this command is used to set the mode to LEARN, the current program will be lost.

Command: **PROG_MODE mode**

Query: **PROG_MODE?**

Response: PROG_MODE mode

Arguments: mode choice of OFF, LEARN, or RUN

Examples:

PROG_MODE LEARN Begins the learn process.

PRMO RUN Begins execution of the current program at the speed specified by the PROG_SETUP SPEED parameter.

PRMO? Requests the current state.
PRMO OFF Reports that the current operating mode for Program is off.

See Also: PROG_SETUP

PROG_RECALL PRRC

Purpose: Recall a program from a disk file.

Command: **PROG_RECALL**

Example: **PRRC** Recalls a program as specified by the current values of the PROG_SETUP parameters DISK and IFILE.

See Also: PROG_SETUP

PROG_SETUP PRSU

Purpose: Set up parameters for the PROG_MODE, PROG_RECALL, and PROG_STORE commands.

Command: **PROG_SETUP keyword,value [,keyword,value ...]**

Query: **PROG_SETUP? keyword,value [,keyword,value ...]**

Response: PROG_SETUP keyword [,keyword ...]

Arguments:

SPEED:	Choice of FAST, MEDIUM, and SLOW.
SLOW	Executes the program one command per second, and displays the text of each command as it is executed.
MEDIUM	Executes the program with a small delay after each command. This gives the display a chance to be updated, so you can see the effect of individual commands. Commands are not displayed.
FAST	Executes the program without any artifical delays, and does not display commands.
DISK:	Choice of INTERNAL and FLOPPY.
INTERNAL	Selects the internal disk as the one used for both the PROG_RECALL and PROG_STORE commands.
FLOPPY	Selects the floppy disk as the one used for both the PROG_RECALL and PROG_STORE commands.
IFILE:	Specifies the name of the program recalled by the next PROG_RECALL command. Choices are determined by the programs on the currently selected disk.

PROG_SETUP (continued) PRSU

OFILE: Specifies the name of the program stored by the next PROG_STORE command. The value should be a valid file name without extension. (One to eight letters and digits, beginning with a letter.)

Examples:

PROG_SETUP SPEED,FAST Sets execution speed to FAST.

PRSU DISK,FLOPPY,IFILE,DEMO Specifies that the next PROG_RECALL command should recall the program DEMO from the floppy disk.

PRSU DISK,INTERNAL,OFILE,TEST Specifies that the next PROG_STORE command should store the program in the file TEST on thei nternal disk.

PRSU? DISK,IFILE,OFILE Requests the current values of the DISK, IFILE, and OFILE parameters.
PROG_SETUP DISK,INTERNAL,IFILE,DEMO,OFILE,TEST

PRSU? Requests all parameters.
PROG_SETUP DISK,INTERNAL,IFILE,DEMO,OFILE,TEST,SPEED,FAST

Notes: If both the DISK and IFILE keywords are given in the same PROG_SETUP command, the DISK keyword should be specified first.

See Also: PROG_MODE, PROG_RECALL, PROG_STORE , PROG_LIST

PROG_STORE PRST

Purpose: Stores a program in a disk file. Any file on the specified disk with the same name will be replaced.

Command: **PROG_STORE**

Example: **PRST** Stores a program as specified by the current values of the PROG_SETUP parameters DISK and OFILE.

See Also: PROG_SETUP

PROTECT_MODE PRMD

Purpose: Turns ON/OFF Protected Mode. When PROTECT_MODE is ON, data will be acquired and stored in non-volatile protected memory. The plug-ins will not re-arm after the data is acquired. When PROTECT_MODE is OFF, the system operates normally.

Command: **PRMD enable**

Query: **PRMD?**

Arguments: enable - ON or OFF

Default: OFF

PRW_ON_STATE PWRO

Purpose: Selects the state of the Trig Enable at power on. When the PWRO_ON_STATE is RESTORED, the Trig Enable will be restored to it's prior to losing power. If the Trig Enable was externally enabled when power was interrupted, the Trig Enable will be internally enabled. When the PWRO_ON_STATE is DEFAULT, the Trig Enable will be waiting for an external enable at power on.

Command: **PWRO state**

Query: **PWRO?**

Response: PWRO state

Argument: state - DEFAULT or RESTORED

QYR?

Purpose: Reads and then clears the contents of the Query error Register (QYR). The QYR queue identifies the most recent query error. A query error occurs when a query or its response is not read by the controller according to the IEEE-488.2 Message Exchange Protocol. For example, if the controller sends a query before it reads the response to the previous query, the previous response is flushed, a query error is logged, and the second response is sent.

The QYR is a queue which contains a unique encoded value. The value corresponds to the most recent query error.

Clearing the QYR register also clears the QYB summary message bit (bit # 2) of the standard Event Status Register (ESR) and the effect could propagate to the main Status Byte (STB).

Query: **QYR?**

Response: QYR value

Argument:

value	Corresponds to an error such as: 7200A is read with no data in output queue data in output queue is lost deadlock: input and output buffers full controller reads before sending a complete query controller sends new command before reading last query

A listing of all encoded values with their meanings is found in Appendix A.

Example:

QYR?	Read and clear the QYR
QYR 2	Response indicates Interrupted Action; that is, the controller sent a command or query without fully reading the response to a previous query.

See Also: ALST?, ESE, *ESR?, *STB?

RECALL REC

Purpose: Performs the waveform recall operation previously configured by RECALL_SETUP.

Command: **RECALL**

See Also: RECALL_SETUP, STORE_SETUP, STORE, DIRECTORY_LIST

RECALL_PANELS RCPN

Purpose: Recalls panel settings from either floppy disk or internal disk.

Command: **RECALL_PANELS keyword,value,keyword,value**

Query: **RECALL_PANELS?**

Response: RECALL_PANELS keyword,value,keyword,value,...keyword,value

Arguments: An argument consists of a keyword followed by its value. Any number of arguments in any order may be used to change individual settings.

Keyword	Meaning:Value	
DISK	Disk from which panel settings will be recalled:	
	INT	internal disk
	FLOPPY	floppy disk
FILE	Filename from which panel settings will be recalled:	
	filename	must not include an extension. The 7200A automatically adds .PNL

Examples:

RCPN DISK,INT,FILE,"MIKE" Recalls panel settings from the file MIKE.PNL on the internal disk.

RCPN? Requests the current defaults for the RCPN command

RCPN DISK,INT,FILE,"PANEL" Reports that the panel settings for file PANEL will be recalled from the internal disk.

Notes: If no arguments are given, the panel is recalled from the file last specified.

See Also: STORE_PANELS, DIRECTORY_LIST

RECALL_SETUP RCST

Purpose: Configures the waveform recall operation.

Command: **RECALL_SETUP keyword,value,keyword,value,...keyword,value**

Query: **RECALL_SETUP?**

Response: RECALL_SETUP keyword,value,keyword,value,...keyword,value

Arguments: An argument consists of a keyword followed by its value. Any number of arguments in any order may be used to change individual settings.

Keyword	Meaning:Value
M1,M2,...M8	Memory to which waveform will be recalled filename.XXX Filename of floppy disk file containing waveform to be recalled to memory.
T1= M1,T2= M2,...T8= M8	Memory and trace to which waveform will be recalled. The waveform will be recalled to the specified memory. The trace equation of the specified trace will be changed to Tx= Mx and the trace will be turned on. filename.XXX Filename of floppy disk file containing waveform to be recalled to memory. Filename extension auto increment OFF All recall operations read the same file. ON For each recall operation, the numeric extension of the previously read file is incremented to the next file on the disk.

RECALL_SETUP (continued) RCST

Examples:

RCST M1,"TRACE1.000",T3= M3,'LASER.014"

Configures the waveform recall operation to:
recall the waveform in file TRACE1.000 to memory M1;
recall the waveform in file LASER.014 to memory M3, change the trace equation of trace 3 to be T3= M3, and turn on trace 3

RCST M1,'",M3,'LASER.000",FINC,ON

Configures the waveform recall operation to:
not recall anything to memory M1;
recall the waveform in file LASER.000 to memory M3
Filename extension autoincrement is enabled so that subsequent recall operations will search for the next file on the disk with filename LASER.XXX.

Note: RECALL_SETUP configures the recall operation but does not initiate it. The actual recall operation does not occur until the RECALL command is issued.

See Also: RECALL, STORE_SETUP, STORE, DIRECTORY_LIST

RECORD_TRACES RECT

Purpose: Turns record traces mode on or off.

Command: **RECORD_TRACES argument**

Query: **RECORD_TRACES?**

Response: RECORD_TRACES argument

Argument: The argument specifies whether to turn record traces mode on or off:

ON
OFF

Examples: **RECT ON** Turns on record traces mode.

See Also: REPLAY_TRACES

REFERENCE_CLOCK RCLK

Purpose: Selects the system clock used as the reference for the plug-in timebases.

Command: **REFERENCE_CLOCK arg**

Query: **REFERENCE_CLOCK?**

Response: REFERENCE_CLOCK clock

Arguments: clock

INT	Selects the 7200A clock.
EXT	Selects an externally applied 10 MHz clock. The external clock signal is applied to the back of the 7200A.

Examples:

REFERENCE_CLOCK EXT	Sets the reference clocks for all plug-ins to the user supplied clock.
REFERENCE_CLOCK?	Queries which clock is being used as a reference.
REFERENCE_CLOCK INT	Reports that the internal reference clock is selected.

REMOTE REM

Purpose: Place the 7200A into Remote mode. In Remote mode the 7200A responds only to remote commands and queries; all front panel keys on the mainframe and plug-in(s) are disabled, except for the Go To Local softkey located in the lower left part of the screen. Pressing this key will restore front panel control.

Command: **REMOTE**

Examples: **REM** Places the 7200A into Remote mode and disables front panel control

Note: The 7200A also switches to Remote mode if it is in Local mode and the computer addresses the 7200A to Listen with REN (Remote Enable) asserted. However, this will only happen if the GPIB interface had been set to the Local state. Pressing the GO To Local key places the 7200A into Local mode but has no effect on the GPIB interface. The GPIB interface is set to Local mode at any time by deasserting REN. Sending the GTL interface command after addressing the 7200A to Listen will also put it into Local mode.

Sending REMOTE will always place the 7200A into Remote mode. If REMOTE is sent with REN asserted, the GPIB interface is put into Remote mode. If REMOTE is sent with REN deasserted, the GPIB interface is put into Local mode.

See IEEE-488.1 (section 2.8) for more details concerning GPIB state transitions.

See Also: LOCAL, LOCKOUT

REM_CTRL RCTL

Purpose:	selects the port (GPIB or RS232) from which the 7200A will accept remote commands.	
Command:	**REM_CTRL port**	
Query:	**REM_CTRL?**	
Response:	REM_CTRL port	
Arguments:	port	either GPIB or RS232
Examples:	**REM_CTRL GPIB**	Sets the remote host port to GPIB
	REM_CTRL?	Queries the 7200A for the current remote host port.
	REM_CTRL RS232	reports that the host port for the 7200A is RS232
Notes:	Care should be used when issuing this command remotely since it immediately changes the host port.	
See Also:	GPIB_ADDRESS, COMM_RS232	

REPLAY_TRACES REPT

Purpose: Provides access to traces previously recorded in Record Traces mode.

Command: **REPLAY_TRACES argument**

Query: **REPLAY_TRACES?**

Response: REPLAY_TRACES argument

Argument:

ON	start replay traces
OFF	stop replay traces
NEXT	step forward in buffer of recorded traces
PREV	step back in buffer of recorded traces

Examples:

REPT ON	Start replay traces.
REPT PREV	Step to previously recorded traces.

Note: Query returns only ON or OFF.

See Also: RECORD_TRACES

SCSI_ID? SCID?

Purpose: Thsis is a query function to return the SCSI id of the 7200A.

Query: **SCID?**

Response: SCID value

Example:

SCID?	Request SCSI id of the 7200A
SCSI_ID 7	Returns current setting of ID

SELECT SEL

Purpose: Selects the trace that is acted upon when performing the various display commands and moves the selection frame to that trace. If the selected trace is not on, an error is reported.

Command: **prefix:SELECT**

Query: **prefix:SELECT? state**

Response: prefix:SELECT

Argument:

prefix	The prefix is limited to traces, i.e., T1, T2, ... T8.	
state	ON	If it is the selected state.
	OFF	If it is not the selected state.

Examples:

T1:SEL	Selects trace 1 as the SELECT'ed trace.
SEL?	Requests the identity of the selected trace.
T1:SEL ON	Reports that trace 1 is the selected trace.

See Also: TRACE, CURSOR_MEASURE

STOP

Purpose: *Forces the end of the current acquisition. Changes the acquisition state from READY to TRIGGERED. If the Trigger Mode is AUTO or NORM, it will change the mode to SINGLE to prevent further acquisition.*

Command: **STOP**

Note: In sequence mode, the acquisition process is halted.

See Also: INR?, INE, *TRG, WAIT

STORE STO

Purpose: Performs the trace storage operation previously configured by STORE_SETUP.

Command: **STORE**

Note: Only those traces which are displayed (and which have been configured by STORE_SETUP) will be stored.

See Also: STORE_SETUP, RECALL_SETUP, RECALL, DIRECTORY_LIST

STORE_PANELS STPN

Purpose: Stores current panel settings to either floppy disk or internal disk.

Command: **STORE_PANELS keyword,value,keyword,value**

Query: **STORE_PANELS?**

Response: STORE_PANELS keyword,value,keyword,value,...keyword,value

Arguments: An argument consists of a keyword followed by its value. Any number of arguments in any order may be used to change individual settings.

Keyword	Meaning:Value
DISK	Disk to which panel settings file will be stored: INT internal disk FLOPPY floppy disk
FILE	Filename to which panel settings will be stored: filename must not include an extension. The 7200A automatically adds .PNL

Examples: **STPN DISK,INT,FILE,"MIKE"** Store panel settings to the file MIKE.PNL on the internal disk.

STPN? STPN DISK, INT, FILE, EXP1"

Note: If no arguments are given, the panel is saved to the file last specified.

See Also: RECALL_PANELS

STORE_SETUP STST

Purpose: Configures the trace storage operation.

Command: **STORE_SETUP keyword,value,keyword,value,...keyword,value**

Query: **STORE_SETUP?**

Response: STORE_SETUP keyword,value,keyword,value,...keyword,value

Arguments: An argument consists of a keyword followed by its value. Any number of arguments in any order may be used to change individual settings.

Keyword	Meaning:Value	
T1,T2,...T8	Trace to be stored	
	DIS	Disable storage of trace.
	FILE	Store trace to floppy disk. Filename is equal to trace label plus filename extension
.	M1,M2,...M8	Store trace to specified memory.
	T1= M1,T2= M2,...T8= M8	Store trace to specified memory and change specified trace equation to Tx= Mx.
FEXT	Filename extension for storing to floppy disk: 000,001,...999	
FINC	Filename extension autoincrement	
	OFF	All trace storage operations overwrite the same file.
	ON	Each trace storage operation creates a new file by incrementing the extension of the previous file.

STORE_SETUP (continued) STST

Examples:

STST T1,FILE,FEXT,027,T2,M1,T5,T7= M7
Configures the trace storage operation to:
store trace 1 to a floppy disk file with filename equal to the trace label plus extension .027;
store trace 2 to memory M1;
store trace 5 to memory M7, change the trace equation of trace 7 to be T7= M7, and turn on trace 7.

STST T1,DIS,T4,FILE,FEXT,000,FINC,ON
Configure the trace storage operation to:
disable storage of trace 1;
store trace 4 to a floppy disk file with filename equal to the trace label plus extension .000;
Filename extension autoincrement is enabled so that subsequent storage operations will increment the filename extension before storing to floppy disk.

Note: STORE_SETUP configures the storage operation but does not initiate it. The actual store operation does not occur until the STORE command is issued.

See Also: STORE, RECALL_SETUP, RECALL, DIRECTORY_LIST, TRACE_LABEL

TEMPLATE? TMPL?

Purpose: Produces a copy of the template which formally describes the various logical entities making up a complete waveform. In particular, the template describes in full detail the variables contained in the descriptor part of a waveform. Refer to Section 3: Waveform Transfer for further information.

Query: **TEMPLATE?**

Response: See Section 3: Waveform Transfer for a sample template listing.

Note: The template is transmitted as an ASCII string.

See Also INSPECT, WAVEFORM

TIMEBASE_LOCK TBLK

Purpose: Forces all plug-ins to use the same timebase. Any changes to the timebase controls on one plug-in will also change the other plug-ins to the same setting. If plug-ins of different capabilities are used, only timebase settings common to all plug-ins can be selected. If any timebase control setting is not common to all plug-ins, the plug-in timebases cannot be locked.

Command: **TIMEBASE_LOCK state**

Query: **TIMEBASE_LOCK?**

Response: TIMEBASE_LOCK state

Arguments:

state	[ON / OFF]	Turn locking feature on or OFF

Examples:

TIMEBASE_LOCK ON	Locks all plug-ins to the timebase setting of the leftmost plug-in.
TIMEBASE_LOCK?	Queries whether the timebases are locked or independent.
TIMEBASE_LOCK OFF	Reports that the timebases are independent.

TRACE TRA

Purpose: Turns on or off the display of the specified trace. When a trace is turned on, it becomes the selected trace.

Command: **prefix:TRACE state**

The prefix is limited to traces, i.e., T1, T2, ... T8.

Query: **prefix: TRACE?**

Response: prefix:TRACE state

Argument:

state	ON	causes the trace to be on the display
	OFF	takes the trace off the display

Examples:

T1:TRA ON	Tells the 7200A to display trace 1.
T2:TRA?	Asks if trace 2 is displayed.
T2:TRA OFF	Response to query indicates that trace 2 is not being displayed.

Notes: When a trace is turned off, all processing for that trace is discontinued unless that trace is used to define another trace (eg. if T3 = T2 + T1 and T1 is turned off, the processing of T1 will continue since it is needed to compute T3).

See Also: DEFINE, SELECT, XY_ASSIGN

TRACE_ANOT TRAA

Purpose: This command allows the user to annotate a trace with up to 10 different text strings. The position of the text can also be specified.

Command: prefix: TRACE_ANOT number,keyword,value[,keyword,value]... [,keyword,value]

Query: prefix: TRACE_ANOT? number

Response: prefix: TRACE_ANOT number,keyword,value,keyword,value, keyword,value,keyword,value

Arguments:

prefix	The prefix is limited to traces, i.e., T1,T2,...T8.
number	The text string that the keyword,value pairs apply to. The number can be 1,2,...10.
keyword	meaning:value
XPOS	horizontal position of the lower left corner of the text string in the range: -1E30 to 1E30
YPOS	vertical position of the lower left corner of the text string in the range: -1E30 to 1E30 followed by the vertical units of the traces.
STATE	selects if text string is displayed: ON or OFF.
TEXT	the string to be displayed. it can be up to 15 characters and must be surrounded by quotes.

Example:

T1:TRAA 2,XPOS,50E-9,YPOS,0.2,STATE,ON,TEXT,
My display the words "My Point" on the point of the grid corresponding to 50 ns and 200mV. It is the second string for trace 1.

T2:TRAA? 3 requests the arguments of the third string of trace 2.

T2:TRAA 3, XPOS,1.75E-, YPOS,1.55,STATE,ON,TEXT,"1'mer"
reports that the string "1'mer" is displayed at the point on the grid that corresponds to 1.75 µs, 1.55V

TRACE_ANOT (continued) TRAA

Note: When a waveform is read out of the 7200A using the WAVEFORM or INSPECT commands, all strings which are displayed are included in the TEXT block of the waveform.

See Also: WAVEFORM, INSPECT

TRACE_LABEL TRLB

Purpose: Defines an application specific trace identifier for annotation and disk storage. If the default label is selected, the trace equation annotates the trace on the Main Screen. Selecting a label other than the default will force the label to annotate the trace on the Main Screen. The trace label is also used as the filename when storing traces to floppy disk.

Command: **prefix: TRACE_LABEL label**

Query: **prefix: TRACE_LABEL?**

Response: prefix: TRACE_LABEL label

Arguments:

prefix The prefix for this command is limited to traces, i.e., T1, T2, ... T8.

label A string argument that must be a valid DOS filename; that is, it cannot begin with a number and cannot be more than 8 characters long.

Default: Trace1 for trace 1, Trace2 for trace 2, and so on

Examples:

T1:TRACE_LABEL Trace1 sets the trace label for trace 1 to Trace1

T8:TRACE_LABEL? requests the trace label for trace 8.

T8:TRACE_LABEL Trace8

Note: Filenames are case insensitive. That is, Trace1 is the same as TRACE1.

See Also: DEFINE, DEFINE_REPLAY, STORE

TRANSFER_FILE TRFI

Purpose: Transfer panel setup and ICL program files between the host computer and the 7200A. The command form transfers a file from the host to the 7200A. The query form transfers a file from the 7200A to the host.

Command: **TRANSFER_FILE disk, file, data**

Query: **TRANSFER_FILE? disk, file**

Arguments:

disk: choice of INT and FLOPPY

INT	Selects the internal disk.
FLOPPY	Selects the floppy disk.

file: Specifies the file to be transferred. Filenames are made of two parts, separated by a dot. The first part is the name, and the second is the extension.

The name is one to eight characters long, and begins with a letter. The remaining characters may be letters, digits, or underscores.

The extension specifies which type of file is being transferred. Choices include:

COL	color scheme files for color display
MON	color schemes for monochrome display
PNL	Panel setup file
SRC	ICL program source file
APD	ICL program data file

data Contains the block format specification followed by the data to be transferred to the file.

Examples: **TRFI INT,"START.PNL",# 9nnnnnnnnn< data>**

creates the ICL panel setup file "START.PNL" on the internal disk.The data are given in the definite length block format. "n" is one of the nine bytes when taken together indicate the total number of bytes of data.

TRANSFER_FILE (continued) TRFI

TRFI FLOPPY,"PROG.SRC",# 0< data> < end>

creates the ICL program source file "PROG.SRC" on the floppy disk. The data are given in the indefinite length block format.

TRFI? INT,'SAVED.PNL"

returns the data from the panel setup file "SAVED.PNL" on the internal disk. The response depends on the parameters set using the communication setup commands (COMM_FORMAT, COMM_HEADER, and COMM_RS232).

Notes: The response to the query explicitly includes the disk and file arguments unless COMM_FORMAT OFF is previously specified.

See Also: COMM_FORMAT, COMM_HEADER, and COMM_RS232

TRIG_ENABLE TREN

Purpose: Selects the state of the Trig Enable. When TRIG_ENABLE is SET, the Trig Enable will be internally enabled and the plug-in will be seeking triggers. When TRIG_ENABLE is CLEAR, the Trig Enable will be reset and be waiting to be externally enabled before the plug-in will trigger.

Command: **TREN state**

Query: **TREN?**

Response: TREN state

Arguments: state - SET or CLEAR

Default: CLEAR

TRIG_LOCK TRLK

Purpose: Locks all plug-ins to one trigger source. The leftmost plug-in will initially be selected as the trigger master. To change the trigger master, set trigger source on slave. When turned OFF, each plug-in will trigger independently based on its own trigger control setting.

Command: **TRIG_LOCK state**

Query: **TRIG_LOCK?**

Response: TRIG_LOCK state

Arguments: state: [ON / OFF] Turn the locking feature ON or OFF.

Examples:

TRIG_LOCK ON Locks all plug-ins to the trigger source of the left most plug-in.

TRIG_LOCK? Queries whether the triggers are locked or independent.

TRIG_LOCK OFF Reports that the triggers are independent.

See Also: TRIG_SELECT in Section 6, plug-in remote commands

Notes: Trigger controls of slave plug-ins can not be modified.

TRIG_MODE TRMD

Purpose: Sets the Trigger Mode for the next acquisition.

Command: **TRIG_MODE mode**

Query: **TRIG_MODE?**

Response: TRIG_MODE mode

Argument:

mode	meaning
AUTO	Auto trigger mode
NORM	Acquires data continuously
SINGLE	Acquires and displays one waveform
SEQueNCE	Acquires and displays one sequence waveform
SEQNORM	Acquires and displays sequence waveforms continuously.

Examples:

TRMD NORM	Sets the trigger mode to NORMAL, or as a response, it indicates the setting as NORM.
TRMD?	Requests the current trigger mode setting
TRMD AUTO	Reports that the trigger mode is set to AUTO.

Note: Sequence Trig Mode cannot be selected when interleaved sampling is on and the current timebase does not allow single shot acquisitions.

Using trigger modes SEQUENCE and SINGLE, this command will not arm the trigger. Use command ARM_AQUISITION to actually start a single or sequence acquisition.

See Also: *TRG, STOP, ARM_ACQUISITION

UPTIME UPTI

Purpose: Report the length of time the scope has been operating since it was turned on or last time is was set to the default state.

The time may be set to 0 using the "Default Settings" softkey in the "Configure System" screen ot the "RST command.

Query: **UPTIME?**

Example:

UPTI?	Requests the running time in milliseconds.
UPTI 5104000	Indicates that the scope has been running for 5104 seconds since power-on or default settings.

VERT_MAGNIFY VMAG

Purpose: Sets the vertical display gain (not the plug-in's gain) of the specified trace. Increasing the vertical gain expands the trace on the screen. Decreasing it attenuates the display of the trace.

Command: **prefix:VERT_MAGNIFY mag**

The prefix is limited to traces, i.e., T1, T2, ... T8.

Query: **prefix:VERT_MAGNIFY?**

Response: prefix:VERT_MAGNIFY mag

Argument: mag Choices for magnification are: 20 (greatest), 10, 5, 2, 1, .5, .2, .1, and .05 (least magnification).

Examples:

T2:VMAG 10 Sets the gain to 10.

T3:VMAG? Requests the vertical magnification of trace 3.
T3:VMAG 10 Reports the magnification gain as 10.

See Also: VERT_POSITION

VERT_POSITION VPOS

Purpose: Sets the vertical display position (not the plug-in's offset) of the specified trace. Increasing the vertical position moves the waveform towards the top of the display while decreasing the value moves it towards the bottom. The trace may be positioned by up to + 69 or -69 divisions, relative to the original position at acquisition. This allows a fully expanded waveform to be positioned anywhere on the screen.

Command: **prefix:VERT_POSITION pos**

The prefix is limited to traces, i.e., T1, T2, ... T8.

Query: **prefix:VERT_POSITION?**

Response: prefix:VERT_POSITION pos

Argument: pos — Position in the range -69.0 to 69.0 in increments of .02.

Examples:

T3:VPOS 1.5 — Offsets trace 3 by 1.5 divison from screen center towards the top of the display.

T2:VPOS? — Requests the vertical position of trace 2.
T2:VPOS 1.0 — Indicates the vertical position for trace 2 as 1.0 division from screen center.

Note: Trace Position = (VPOS) * (VMAG) Divisions from original acquisition offset.

For example:

If VMAG = 1
TA:VPOS 3 Will position the A trace 3 divisions above the origin

If VMAG = 2
TA:VPOS 3 Will position the A trace 6 divisions above the origin

See Also: GRID, VERT_MAGNIFY

WAIT

Purpose: Prevents the 7200A command parser from executing new commands until the oscilloscope has completed the current acquisition process.

Command: **WAIT timeout**

Argument: timeout range of 0...10,000s in increments of 10 ms.
A timeout of 0 will wait forever.

Example: The following example finds the maximum amplitude (using the command, T1:PAVA? MAX) of several different sweeps of Trace 1.

First, send TRMD SINGLE which sets the Trigger Mode to SINGLE.

```
loop { send : ARM; WAIT 0; T1:PAVA? MAX
    read response
    process response
     }
```

The WAIT command ensures that the maximum is evaluated only after a new waveform is acquired.

Note: The WAIT command can only be aborted using DEVICE CLEAR.

WAVEFORM WF

Purpose: Transfers a waveform from the host computer to the 7200A.

Command: **prefix: WAVEFORM ALL, waveform**

Arguments:

prefix — The prefix for this command is limited to memories i.e., M1, M2, ... M8.

waveform — contains the block format specification followed by all the parts of the waveform, i.e., waveform descriptor, data block(s), etc.

Examples:

M2: WF ALL, # 9nnnnnnnnn< descriptor><text><time><dat1><dat2> transfers to the 7200A all the blocks of a waveform into memory 2. Use the definite length block format.

n is one of the nine bytes which when taken together, indicate the total number of bytes in the waveform, i.e., bytes in the descriptor plus those in the text, etc.

< text> and < time> represent annotations and RIS times, respectively. Each may be optionally included.

< dat1> , < dat2> , or both may be used for the waveform data blocks. If both are used, their order must correspond to that indicated in the descriptor.

M2: WF ALL, # 0 < descriptor> < text> < time> < dat1> < end> transfer to the 7200A all the blocks of an unprocessed waveform into memory 2. For GPIB < end> is line feed with asserted EOI line or just asserted EOI. For RS-232-C, < end> is defined by the End_In argument of the COMM_RS232 command.

Note: The parameters set by the communication commands are specified within the waveform descriptor. These settings are used when waveforms are transferred out of the 7200A. Only # 9 and # 0 formats are allowed for transferring data into the 7200A. OFF format is not allowed.

See Also: WAVEFORM?, COMM_RS232

WAVEFORM? WF?

Purpose: Transfers a waveform, or part of a waveform, from the 7200A to the host computer.

Query: **prefix: WAVEFORM? arg**

Arguments: prefix The prefix for this command can be channels or traces ! i.e., T1, T2, ... T8, A1, A2, B1, B2.

arg choice of:

ALL	all the waveform blocks
DESC	waveform descriptor(acquisition settings)
TEXT	user defined trace annotation
TIME	RIS time descriptor
DAT1, DAT2	1st and 2nd blocks of waveform data

Example: **T2: WF? DAT1** transfer a block of data points from trace 2 to the host computer. The 7200A response depends on the parameters set using the communication commands (COMM_FORMAT, COMM_ORDER, COMM_HEADER, and COMM_RS232).

Notes: If no argument is specified, the 7200A responds as if ALL is sent.

The response to the query explicitly includes the argument sent unless COMM_FORMAT OFF is previously specified.

Although the ALL seems excessive, it allows the possibility for transferring the data back to the oscilloscope. That is, only waveforms having a descriptor and all the other waveform blocks listed in the descriptor can be written back into the 7200A.

Data read from the oscilloscope using this command can be transferred back to the oscilloscope only if COMM_FORMAT is DEF9 or IND0.

See Section 3: Waveform Transfer for an interpretation of the different parts of the waveform response.

See Also: WAVEFORM, COMM_FORMAT, COMM_ORDER, COMM_HEADER, COMM_RS232, TRACE_ANOT

WPE

Purpose: Sets the Waveform Processing event Enable register (WPE). The WPE register determines which events in the Waveform Processing status Register (WPR) are reported. WPR identifies which trace has been processed completely. Any reported WPR event sets the WPB summary message bit (bit # 1) of the Data Processing Register (DPR) and propagates to the main Status Byte (STB).

Command: **WPE mask**

Query: **WPE?**

Response: WPE mask

Argument: mask When expressed in binary, this number (between 0 and 255) represents the bits of the WPR that can be reported:

Bit #	Significance
7	Waveform processing for Trace 8 done
6	Waveform processing for Trace 7 done
5	Waveform processing for Trace 6 done
4	Waveform processing for Trace 5 done
3	Waveform processing for Trace 4 done
2	Waveform processing for Trace 3 done
1	Waveform processing for Trace 2 done
0	Waveform processing for Trace 1 done

Examples: **WPE 3** Set bit 1, i.e., decimal 2, and set bit 0, i.e. decimal 1. Summing these two values yields the WPE mask of 2+ 1 = 3.

WPE? Requests the contents of the WPE.
WPE 3 Response indicates that events for traces 1 and 2 can be reported.

See Also: DPE, DPR?, WPR?

WPR?

Purpose: Reads and then clears the Waveform Processing status Register (WPR). WPR signals that processing on a particular waveform has completed. For history functions (Average, Histogram, Extrema,...), a completed signal may correspond to a partial result. That is, the waveform processing done bit is set when a trace can be displayed. For history functions, traces can be displayed before the maximum sweeps have accumulated.

Clearing the WPR register also clears the WPB summary message bit (bit # 1) of the Data Processing Register (DPR) and the effect could propagate to the main Status Byte (STB).

Query: **WPR?**

Response: WPR value

Argument: value When expressed in binary, this number (between 0 and 255) represents the bits of the WPR:

Bit #	Significance
7	Waveform processing for Trace 8 done
6	Waveform processing for Trace 7 done
5	Waveform processing for Trace 6 done
4	Waveform processing for Trace 5 done
3	Waveform processing for Trace 4 done
2	Waveform processing for Trace 3 done
1	Waveform processing for Trace 2 done
0	Waveform processing for Trace 1 done

Examples: **WPR?** Read and then clear the WPR.
WPR 3 Response indicates that waveform processing is done for traces 1 and 2.

See Also: ALST?, DPE, DPR?, *STB?, WPE

XY_ASSIGN XYAS

Purpose:	Assigns traces to the X and Y axis to create an X versus Y display.	
Command:	**XY_ASSIGN X trace, Y trace**	
Query:	**XY_ASSIGN?**	
Response:	XY_ASSIGN X trace, Y trace	
Argument:	X trace, Y trace	trace is limited to T1, T2,...,T8
Examples:	**XYAS T1,T2**	Assigns trace 1 as the X axis and trace 2 as the Y axis
	XYAS?	Returns which traces are assigned to the X and Y axes.
	XYAS T4,T3	Indicates that trace 4 is assigned to X and trace 3 to Y.
Note:	X and Y trace must be different.	
	Turning traces on or off may change which traces are assigned.	
See Also:	TRACE, XY_DISPLAY	

XY_CURSOR_ORIGIN XYCO

PURPOSE: The XY_CURSOR_ORIGIN command sets the position of the origin for absolute time cursor measurements on the XY display.

Absolute time cursor values may be measured either with respect to the point (0,0) volts (OFF) or with respect to the center of the XY grid (ON).

The XY_CURSOR_ORIGIN query returns the current assignment of the origin for absolute time cursor measurements.

Command: **XY_CURSOR_ORIGIN mode**

Query: **XY_CURSOR_ORIGIN?**

Response: XYCO mode

Arguments:

mode	meaning
ON	Horizontal Absolute cursor values are calculated with respect to center of the grid.
OFF	Horizontal Absolute cursor values are calculated with respect to 0 volts.

Examples:

XYCO ON	Sets the origin of the Horizontal Absolute cursors to the center of the grid
XYCO?	Requests current origin for the Horizontal Absolute cursors.
XYCO OFF	Indicates that the origin for the Horizontal Absolute cursor is 0 volts

See Also: XY_CURSOR_VALUE, CURSOR_MEASURE

XY_CURSOR_SET XYCS

Purpose: The XY_CURSOR_SET command allows the user to position any one of the nine independent XY cursors at a given screen location. The positions of the cursors can be modified or queried even if the required cursor is not currently displayed or if the XY display mode is OFF.
The XY_CURSOR_SET? query indicates the current position of the cursor(s).

Command: **XY_CURSOR_SET keyword,position, ...,keyword, position**

Query: **XY_CURSOR_SET? keyword,...,keyword**

Response: XYCS keyword,position,...,keyword,position

Arguments: keyword,position,...,keyword,position

Cursor Type	keyword	position
Marker	HABS	0 to 10 DIV
Horizontal	HREF, HDIF	0 to 10 DIV
Vertical Absolute X Axis	XABS	-4 to 4 DIV
Vertical Relative X Axis	XREF, XDIF	-4 to 4 DIV
Vertical Absolute Y Axis	YABS	-4 to 4 DIV
Vertical Relative Y Axis	YREF, YDIF	-4 to 4 DIV

The four cursor types measure the following:

Vertical Absolute — measures the absolute vertical value at a given point.

Vertical Relative — measures the difference between vertical positions of the cursors of a trace(s).

Marker — measures the absolute horizontal position and its vertical value of a point on a trace(s).

Horizontal — measures the difference between the horizontal positions andtheir corresponding vertical values of two Horizontal cursors.

XY_CURSOR_SET (continued) XYCS

Examples:

XYCS XREF,3DIV,XDIF,-2DIV Places the XREF cursor at + 3DIV and the DIF at -2DIV

Notes:

With query, if no keyword is specified, the positions of all cursors are returned.

The units division is optional.

Arguments are grouped in pairs. The first one names the cursor to be modified and the second one indicates its new value. Pairs may be given in any order.

See Also:

XY_CURSOR_ORIGIN, XY_CURSOR_VALUE?

XY_CURSOR_VALUE? XYCV?

Purpose: Returns the current values of the X versus Y trace parameters for a given cursor type. The cursors need not be displayed to obtain these parameters.

Query: **XY_CURSOR_VALUE? keyword,...,keyword**

Response: XY_CURSOR_VALUE keyword,value,...,keyword,value

Argument: value indicates decimal value followed by the units of the parameters.

For each cursor type, there are six parameters. They are:

keyword	value returned
cursor type_X	value of x at cursor
cursor type_Y	value of y at cursor
cursor type_RATIO	the ratio of delta y to delta x
cursor type_PROD	the product of delta y and delta x
cursor type_ANGLE	the polar co-ordinate angle (theta) of delta y with respect to delta x
cursor type_RADIUS	the distance to the origin

The four cursor types are:

cursor type		meaning
VABS	Vertical Absolute	measures the absolute vertical value at a given point
VREL	Vertical Relative	measures the difference between the vertical positions of two cursors
HABS	Marker	measures the absolute horizontal position and its vertical value of a point on a trace(s).
HREL	Horizontal	measures the difference between the horizontal positions and their corresponding vertical values of two horizontal cursors

XY_CURSOR_VALUE?(continued) XYDS

Examples:

XYCV? HREL_RATIO	Requests the ratio of the values of the y axis to the x axis using the Horizontal Relative cursor
HREL_RATIO,.5	Indicates that the ratio of the x and y values at the Horizontal Reference and Difference cursors is .5.
XYCV? HREL_PROD	Requests the products of the values of the x and y axis using the Horizontal Relative cursors.
HREL_PROD 2V**2	Indicates that the product of the values of the x and y axis at the Horizontal Reference and Difference cursors is 2 Volts squared.

Notes:

For HREL and VREL, delta x and delta y are the difference between the two cursors.

For HABS and VABS, delta x and delta y are the difference between the origin and the cursor.

If the parameter is not specified or equals ALL, all the measured cursor values are returned. If the value of a cursor could not be determined in the current environment, the value UNDEF will be returned.

See Also: XY_CURSOR_SET, XY_CURSOR_ORIGIN

XY_DISPLAY XYDS

Purpose: Enables or disables the XY display mode. The X versus Y (XY) display mode allows you to display one trace against another. When enabled, a square grid, in the top half of the screen is used for the XY display while the rectangular grid underneath simultaneously shows the original source waveforms.

For the XY display to be correctly generated, the traces selected must be of the same time or frequency interval/point and have the same horizontal units (seconds or hertz). As soon as two compatible traces have been selected, the XY display is automatically generated.

Command: **XY_DISPLAY**

Query: **XY_DISPLAY?**

Response: XY_DISPLAY mode

Argument:

mode	ON	causes the XY display to be on
	OFF	causes the XY display to be off

Examples:

XYDS ON	Turns on the XY display mode.
XYDS?	Returns the mode of the XY display.
XYDS OFF	Indicates that the XY display is currently off.

Notes: If persistence mode is turned on when the XY display is active, successive sweeps will be overlaid in the X vs Y display.

See also: XY_ASSIGN, PERSIST, PERSIST_SETUP

*CAL ?

Purpose: Performs an internal calibration and reports results. All input channels are calibrated for the current gain, bandwidth, and timebase settings.

At the end of calibration, the response to *CAL ? indicates how calibration terminated. After calibration, the instrument returns to the state it was in prior to the query

Query: ***CAL?**

Response: *CAL diagnostics

Argument: diagnostics Indicates calibration results as follows:

0 Calibration Successful
1 Calibration Failed

Examples: ***CAL 1** Response is set to one, indicating calibration for one or more of the plug-ins has failed.

***CAL?** Requests a calibration and its results
*CAL 0 Reports that calibration was successful for all plug-ins

Note: This calibration is performed periodically unless otherwise disabled.

See Also: AUTO_CAL, EAR?, EBR?

*CLS

Purpose:	Clears all status registers summarized in the main Status Byte (STB).
Command:	***CLS**
Note:	*CLS is a mandatory IEEE-488.2 Command. It should be sent before monitoring any status register to clear any previously set state.
See Also:	All commands that clear the individual status bytes.

*ESE

Purpose: Sets the bits of the standard Event Status Enable register (ESE). The ESE register determines which events in the standard Event Status Register (ESR) are reported. The ESR reports errors and events that are useful for synchronization. Any reported ESR event sets the ESB summary message bit (bit # 5) of the main Status Byte (STB). The bits in the ESE register have been defined by IEEE-488.2.

Command: ***ESE mask**

Query: ***ESE?**

Response: *ESE mask

Argument: mask When expressed in binary, this number (between 0 and 255) represents the bits of the ESR register that can be reported:

Bit #	Significance
7	1= Powered On
6	User Request
5	Command Error (syntax,invalid cmd,...)
4	Execution Error (invalid parameter,...)
3	Device Dependent Error (CAL error,...)
2	Query Error
1	Request Control
0	Operation Complete

Example: ***ESE # HF0** Enables the events represented by the upper four bits of this event register (i.e., PON, SKR, CMR, EXR)

***ESE?** Read the contents of the *ESE
*ESE 240 Response to query, i.e., # HF0 in hexadecimal, indicates the upper four enable bits are set.

Note: *ESE is a mandatory IEEE-488.2 Command.

See Also: *ESR?

*ESR?

Purpose: Reads and then clears the contents of the standard Event Status Register (ESR).

IEEE-488.2 defines the ESR to report error conditions common to most automatic test equipment. The 7200A implements most but not all of these bits for synchronization and error reporting.

Each bit in the ESR is a summary message bit for a status register. The ESR will not clear a bit if its corresponding register is not cleared.

Clearing the ESR register also clears the ESB summary message bit (bit # 5) of the main Status Byte (STB).

Query: ***ESR?**

Response: *ESR value

Argument: value — When expressed in binary, this number (between 0 and 255) represents the bits of the ESR:

Bit #	Associated Status Byte	Significance
7	PON	1= Powered On
6	none	User Request
5	CMR	Command Error (syntax,invalid cmd,...)
4	EXR	Execution Error (invalid parameter,...)
3	DDR	Device Dependent Error (CAL error,...)
2	QYR	Query Error
1	none	Request Control
0	none	Operation Complete

Examples: ***ESR?** Read and then clear the ESR.

***ESR 32** Response, i.e., 20 in hexadecimal, indicates a Command Error occurred.

Note: *ESR? is a mandatory IEEE-488.2 Command. The bits in the *ESR register are defined by IEEE-488.2.

See Also: *ESE

*IDN?

Purpose: Identifies the instrument. The response indicates the manufacturer, the oscilloscope model, the serial number, and the software revision level.

Query: ***IDN?**

Response: *IDN manufacturer, model number,serial number,software revision

Arguments:

model number	Indicates, the model number (7200A)
serial number	Indicates number specific to each instrument
software revision	Indicates the latest update of the instrument's software

All arguments are in ASCII.

Example:

***IDN?** Requests the identity of the instrument
*IDN LECROY,7200A,A12036,1.0.1
Response indicates the manufacturer as LeCroy, the model of the instrument as 7200A, the serial number as an ASCII string, and the software revision.

*LRN?

Purpose:	Returns a command string which contains all the commands necessary to recreate the current panel settings.
Query:	***LRN?**
Response:	*LRN command,value,...,command,value
Notes:	The command string returned is in Remote Command syntax which can be resent to the 7200A.

*OPC

Purpose: To conform to IEEE-488.2, the 7200A will set the OPC bit (bit # 0) in the standard Event Status Register (ESR) in response to an *OPC command. This bit is set upon completion of any operation.

Several operations can be performed at any one time and can be in various states of completion.

To correctly determine which operation completed, monitor the Internal State Register (INR) and the Data Processing Register (DPR).

Query: ***OPC?**

Response: *OPC 1

Notes: The ASCII character "1" is always returned.

*OPC is a mandatory IEEE-488.2 Command.

See Also: DPE, DPR?, INE, INR?, *WAI

*OPT?

Purpose: Identifies reportable oscilloscope options such as processor type, memory size, software options, plug-ins installed, or hardware options. The response indicates all the installed options.

Query: ***OPT?**

Response: *OPT proc, mem, disp, io, softmainframe,options prefix:model# ,..., prefix:model#

Mainframe Options are:

proc — processor option; PR1 (fast processor) or PR2 (medium speed)

mem — memory options; choices are: ME1, (240 MByte hard disk, 2.88 MByte floppy) or ME2 (52 MByte hard disk, 1.44 MByte floppy)

disp — display option; choices are: DS1 (color) or DS2 (monochrome)

io — interface options; may be any or all:
IF1 - IEEE 488 (GPIB)

soft — Software Options; may be any or all:
SW1 - Time Domain Signal Processing package
SW2 - Statistcal Domain Signal Processing package
SW3 - Frequency Domain Signal Processing package

Arguments:

prefix — Prefix A or B specifies the plug-in. Mainframe options have no prefix.

model# — Indicates, in ASCII, the model numbers of the installed options. Multiple options are separated by commas.

Examples:

***OPT?**

*OPT PR1,ME1,DS1,IF1,SW1,SW2,SW3 A:7242,B:7242

Indicates the high speed processor, large memory, color display, GPIB, and all software options are present and that the model numbers of the plug-ins that are installed. The mainframe has a 7242 in plug-in A and plug-in B contains another 7242.

*RST

Purpose: Sets all 7200A and plug-in control settings to their default values.

Command: ***RST**

Example: ***RST** Initiates a soft reset.

Note: *RST has the same effect as pressing the default settings softkey from the Configure System screen.

*SRE

Purpose: Sets the 8-bit Service Request Enable mask (SRE). The SRE mask determines which events in the main Status Byte (STB) register are reported. If an event is reported, an interrupt (SRQ) is sent to the GPIB controller (if used). Clearing the SRE mask disables SRQ interrupts.

Command: ***SRE mask**

Query: ***SRE?**

Response: *SRE mask

The response returns a value which corresponds to the binary sum of all SRE register bits. Note that bit # 6 cannot be set and its returned value is always zero.

Argument: mask — When expressed in binary, this number (between 0 and 255) represents the bits of the STB register that can be reported by an SRQ:

Bit #	Significance
7	(MSB) Program Running Bit
6	RQS (service request) Bit
5	Standard Event Status Bit
4	Message Available bit
3	Reserved
2	Value Adapted Bit
1	Data Processing Bit
0	(LSB) Internal State Change Bit

Examples:

***SRE 17** This command allows an SRQ to be generated as soon as bit # 4 (i.e., decimal 16) and/or bit # 0 (i.e. decimal 1) in the STB register are set. Summing these two values yields the SRE mask 16 + 1 = 17.

***SRE?** Read the contents of the SRE.
*SRE 2 Response indicates that the Data Processing enable bit is set.

Note: *SRE? is a mandatory IEEE-488.2 Command.

See Also: *STB?

*STB?

Purpose: Reads the contents of the main Status Byte register (STB).

Query: ***STB?**

Response: *STB value

Argument: value When expressed in binary, this number (between 0 and 255) represents the bits of the STB:

Bit #	Associated Status Byte	Significance
7 (MSB)	none	Program Running Bit
6	none	RQS (service request) Bit
5	ESR	Standard Event Status Bit
4	(Output)	Message Available bit*
3	none	Reserved
2	none	Value Adapted Bit**
1	DPR	Data Processing Bit
0 (LSB)	INR	Internal State Change Bit

* The Message Available (MAV) bit is set if the output queue is not empty. It informs the system controller that there is still data to output. It is reset once the output queue is empty, indicating that the system controller has read the data from the 7200A. This bit is not affected by the *CLS command.

** The Value Adapted Bit is set to 1 if a received numerical argument was altered before being used in a computation. For example, the 7200A receives A1:TDIV 11ns. Since the timebase can only be set in multiples of 1,2, and 5, the 11ns would get rounded to 10ns. The Value Adapted bit would be set to report that the received value was altered.

Examples: ***STB?** Read contents of the STB
*STB 16 The response indicates that the output queue is not empty.

Note: *STB? is a mandatory IEEE-488.2 Command. STB is defined by IEEE-488.1.

See Also: *CLS, DPE, DPR?, *ESE, *ESR?, INE, INR?, *SRE

*TRG

Purpose: Enables the signal acquisition by changing the acquisition state from TRIGGERED to READY.

Command: ***TRG**

Note: *TRG command, required by IEEE-488.2 is the equivalent of the ARM_AQUISITION and the IEEE-488.1 GET message.

See Also: INR?, INE, STOP, WAIT, ARM_ACQUISITION

*TST?

Purpose: Initiates an internal self-test and reports results. Acquisition channels, time-base, trigger circuits, and other hardware are also tested.

Specific hardware failures are indicated by bits set in the returned status number. If the number is zero (0), no tests failed.

Query: ***TST?**

Response: *TST status

Argument: Status indicates which test failed as follows:

Bit #	Associated Significance
6	Test Result for Mainframe
1	Test Result for plug-in B
0	Test Result for plug-in A

If the test is successful, the bit is cleared, otherwise it is set to one.

Examples:

***TST?** Initiates self-test and requests results.

*TST 1 Response has bit # 0 set to one, indicating that testing for plug-in placed in slot A has failed. The others have passed or no plug-ins are in the slots.

*WAI

Purpose: Required by the IEEE-488.2 standard, has no effect on the 7200A because it starts processing a command when the previous command has been entirely executed.

Command: ***WAI**

See Also: *OPC

LeCroy 7242 Series / 7291 Adapter: Plug-in Remote Commands

Organization

Each command description begins a new page. A command's name (header) is printed in long and short form near the top of the page. Although the long form is used in the description, the short form can be used instead. Below the header are up to eight sections which describe it:

Purpose:	explains a command's use
Command:	contains a command's syntax
Query:	contains the query syntax
Response:	contains the syntax of the response to a query
Argument:	defines a choice(s)
Example:	includes the command being used
Note:	reports additional considerations
See Also:	cites other relevant commands

Command Execution

Execution of program messages depends on the instrument status. As a rule, commands and queries can be executed in either Local or Remote mode.

Before attempting to execute a command or query, the parser scans it to verify its correctness and that sufficient information is given to perform a requested action.

Note

The LeCroy Model 7242A Plug-in Module is an enhanced version of the LeCroy Model 7242 providing increased analog bandwidth specifications and effective bit performance to 500 MHz.

All other specifications, front panel operation, and remote control operation are identical to the LeCroy Model 7242.

Refer to the LeCroy 7242 Series Plug-in Module Operator's Manual for details regarding performance specifications. Refer to the LeCroy Model 7242 Series Plug-in and 7291 2GS/s Adapter Remote Programmer's Manual for information on remote control commands.

ART_REJECT (7242B only) AREJ

Purpose: To turn artifact rejection on or off while in Synchronous Averaging. When enabled, any waveform with an underflow or overflow value is not included in the average. If artifact reject is turned off, any overflows are set to the maximum possible value of the ADC and any underflows to the minimum value.

Command: **prefix:ART_REJECT state**

Query: **prefix:ART_REJECT?**

Response: prefix:ART_REJECT state

Arguments: plugin Prefix A or B specifies the plug-in.

state ON Turns artifact rejection on.

OFF Turns artifact rejection off.

Examples:

A:ART_REJECT ON Sets artifact rejection on.

A:ART_REJECT? Reports the state of artifact
A:ART_REJECT ON as on.

See Also: SEQ_OPTION, SYNC_AVG_OPT, SWEEPS

ATTENUATION ATTN

Purpose: Sets the multiplier to the total vertical gain. If attenuation-coded probes are used, the probe coding contact rings surrounding the CH1, CH2, BNC connectors recognize the attenuation factors of the probes, and this value cannot be modified with this command. If no probe multiplier is detected, this command can be used to set the attenuation reflecting the attenuation for the probe being used. The current value can always be queried.

Command: **prefix:ATTENUATION atten**

Query: **prefix:ATTENUATION?**

Response: prefix:ATTENUATION atten

Arguments:

prefix	Prefix A or B specifies the plug-in followed by the channel number
atten	range 1, 10, 100, 1000

Examples:

A2:ATTENUATION 10	Sets probe attenuation for channel 2 on plug-in A to 10x.
A1:ATTENUATION? A1:ATTENUATION 1	Reports current probe attenuation for channel 1 on plugin A to be 1x.

Notes: A different probe attenuation can be selected for each channel.

The locking of vertical channel controls can only occur when the probe attenuations are equal.

See Also: VOLT_DIV

BANDWIDTH_LIMIT BWL

Purpose: Reduces the bandwidth to 95 MHz. Use it to reduce signal and system noise, or to prevent high-frequency aliasing for sample rates above 200 MS/sec.

Command: **prefix:BANDWIDTH_LIMIT state**

Query: **prefix:BANDWIDTH_LIMIT?**

Response: prefix:BANDWIDTH_LIMIT state

Arguments:

prefix	Prefix A or B specifies the plug-in followed by the channel number
state	Choices are ON or OFF

Examples:

A1:BANDWIDTH_LIMIT ON	Turns on bandwidth limit on plug-in A, channel 1.
B2:BANDWIDTH_LIMIT? B2:BANDWIDTH_LIMIT OFF	Reports current setting for plug-in B, channel 2 to be off.

Note: Bandwidth limit can be adjusted for each channel.

COUPLING CPL

Purpose: Sets the vertical coupling used to couple a signal to the vertical amplifier input. In the AC position, signals are coupled capacitively, thus blocking the inputs signal's DC component and limiting the lower signal frequencies to greater than 10 Hz. The input impedance is 1MΩ. In the DC position, all signal frequency components are allowed to pass through, and the impedance is selectable as 1 MΩ or 50Ω. If the 50Ω input is selected, the signal will be automatically disconnected from the amplifier whenever the maximum dissipation is exceeded. If this condition exists, the input coupling is switched to GND. To clear the overload condition, remove the signal from the input and reselect the desired coupling.

Command: **prefix:COUPLING couple**

Query: **prefix:COUPLING?**

Response: prefix:COUPLING couple

Arguments:

prefix	Prefix A or B specifies the plug-in followed by the channel number	
couple	A1M	AC coupling
	D1M	DC coupling with impedance at 1 MΩ
	GND	Signal is set to ground
	D50	DC coupling with impedance at 50 Ω

Examples:

A1:COUPLING A1M	Turns on AC coupling on plug-in A, channel 1.
A2:COUPLING?	Reports current coupling value for plug-in A,
A2:COUPLING D50	channel 2, to be DC coupling with impedance at 50 Ω.

Note: Coupling can be modified for each channel.

ENHANCED_RES ERES

Purpose: Selects the amount of digital filtering on a signal. By limiting the system bandwidth, noise can be filtered and reduced. Thus, the effective resolution can actually be improved beyond the ADC's ideal performance. Selecting greater values of this parameter increasingly filters noise. Since the ADCs each have 8 bits, selecting the lowest value removes the filter.

Command: **prefix:ENHANCED_RES res**

Query: **prefix:ENHANCED_RES?**

Response: prefix:ENHANCED_RES res

Arguments:

prefix	Prefix A or B specifies the plug-in followed by the channel number
res	Range 8.0 ... 11.0 in increments of .5.

Examples:

A2:ENHANCED_RES 10.0	Sets enhanced resolution on plug-in A , Channel 2
A2:ENHANCED_RES? A2:ENHANCED_RES 9.5	Reports current enhanced resolution on plug-in A, Channel 2 to be 9.5.

Note: Enhanced resolution can be selected for each channel.

The enhanced resolution filter will not be applied while in Sequence Trigger Mode.

FILTER_COEFF (7242B-S1 option only) FCFF

Command Name **prefix:FILTER_COEFF**

Alias prefix: FCFF

Format FCFF?

Purpose This is a query function to read filter coefficients. Returned will be number of filters and for each filter, the number of coefficients followed by the coefficients. All values are decimal integers.

The order of the filters is flash A (0) followed by flash B (1) followed by flash C (2) followed by flash D (3). When the 7291 is attached, the order of the filters is channel 1's filters followed by channel 2:

A1(0) B1(1) C1(2) D1(3) A2(4) B2(5) C2(6) D2(7)

The coefficients themselves are 16 bit signed integers where 16384 (0x4000)=1, 0=0, -16384 (0xC000)=-1

Examples If each coefficient is represented as h[x], then if the sampling rate is 2 GS/s, then the response is:

```
8,   13, h[0],h[1],..., h[12],   <for channel 1 flash A>
     13, h[0],h[1],..., h[12],   <for channel 1 flash B>
     13, h[0],h[1],..., h[12],   <for channel 1 flash C>
     13, h[0],h[1],..., h[12],   <for channel 1 flash D>
     13, h[0],h[1],..., h[12],   <for channel 2 flash A>
     13, h[0],h[1],..., h[12],   <for channel 2 flash B>
     13, h[0],h[1],..., h[12],   <for channel 2 flash C>
     13, h[0],h[1],..., h[12],   <for channel 2 flash D>
     62, h[0],h[1],..., h[61],   <2 GS/s filter>
```

If the sampling rate is 1 GS/s, the response is:

```
4,   13, h[0],h[1],..., h[12],   <for flash A>
     13, h[0],h[1],..., h[12],   <for flash B>
     13, h[0],h[1],..., h[12],   <for flash C>
     13, h[0],h[1],..., h[12],   <for flash D>
```

FILTER_COEFF (continued) FCFF

If the sampling rate is 400 MS/s, the response is:

2, 7, h[0],h[1],..., h[6], <for flash A>
7, h[0],h[1],..., h[6], <for flash C>

If the sampling rate is 200 MS/s or less, the response is:

1, 2, h[0],h[1],..., h[1], <for flash A>

Arguments prefix - plugin (A or B) followed by channel

FILTER_DATA (7242B-S1 option only) FLTD

Command: **plugin:FILTER_DATA filter**

Purpose: Turns ON/OFF correction filters. When FILTER_DATA is ON, acquisition data is first copied to BRAM and then corrected. Data cannot be readout until it is corrected. When FILTER_DATA is OFF, uncorrected data is available within 50 msec after acquisition is complete. Data will not be corrected until all data is transferred into BRAM. Only the first 16380 uncorrected bytes can be read for this plug-in and this uncorrected data can only be read over SCSI. This command must be sent prior to entering Protected Mode; otherwise, the data will not be available to be read over SCSI. If operating in 2 channel mode, then all of channel 1 data will be sent followed by all of channel 2 data, up to 16380 points.

Arguments: plugin - A or B
filter - ON or OFF

Default: ON

Examples:

B:FLTD OFF	Allows the first 16380 bytes of uncorrected data to be available for readout over SCSI.
B:FLTD?	
B:FLTD OFF	Response to query indicates that uncorrected data will be sent out SCSI.

HF_SYNC HFSY

Purpose: Selects whether the trigger source rate is divided. Set HF Sync to ON to allow stable triggering for sources greater than 200 MHz. This control can be used only when SMART TRIGGER™ is not selected.

Command: **prefix:HF_SYNC state**

Query: **prefix:HF_SYNC?**

Response: prefix:HF_SYNC state

Arguments:

prefix		Prefix A or B specifies the plug-in followed by the Trigger source to be modified - 1, 2, 3 EX 10
state	ON	Sets high frequency sync ON. Trigger rate will be divided .
	OFF	Sets high frequency sync OFF.

Examples:

A1:HF_SYNC ON Enables high freq sync for plug-in A when the trigger source is channel 1.

A3:HF_SYNC?
A3:HF_SYNC OFF Reports HFSY for plug-in A is disabled when trigger source is external.

Note: HF_SYNC can be selected for each trigger source.

See Also: TRIG_SELECT

INTERLEAVED ILVD

Purpose: Enables or disables Random Interleaved Sampling (RIS) which has a 50ps per sample resolution, or an equivalent sample rate of 20GS/s. RIS can be enabled under the following conditions:

Max Memory Selection	RIS Timebase Ranges
1k	200ps/div to 250ns/div
2k	200ps/div to 10ns/div
5k	200ps/div to 20ns/div
10k	200ps/div to 50ns/div
20k	200ps/div to 100ns/div
50k	200ps/div to 200ns/div
100k	200ps/div to 500ns/div
200k	200ps/div to 1µs/div
*500k	200ps/div to 2µs/div

*** Available only when 7242B is purchased with memory Option-L1**

Command: **prefix:INTERLEAVED state**

Query: **prefix:INTERLEAVED?**

Response: prefix:INTERLEAVED state

Arguments:

prefix	Prefix A or B specifies the plug-in	
state	RANDOM	creates a RIS record with non-uniform sampling intervals.
	INTERPOLATED	creates a RIS record with uniform sampling intervals
	OFF	Turn RIS off

Examples:

A:INTERLEAVED RANDOM Turns interleaved sampling ON for plug-in A.

A:INTERLEAVED OFF Turns interleaved sampling OFF if timebase is greater than 10nsec/div.

A:INTERLEAVED?
A:INTERLEAVED RANDOM

Reports current value for plug-in A is interleaved sampling RANDOM.

INTERLEAVED (continued) ILVD

Note: The interpolated sampling option uses a linear interpolation algorithm and the nearest neighboring samples to create RIS record with uniform sampling intervals. The random sampling option does not perform this interpolation.

See Also: TIME_DIV

MEMORY_SIZE MSIZ

Purpose: Sets the maximum number of sample points to represent each waveform. By adjusting the maximum memory size, you can tradeoff record length for update rate. Acquire 2k samples to achieve maximum waveform throughput. Obtaining longer records will provide a greater time window of the signal but require more time to process the additional points.

Command: **prefix:MEMORY_SIZE size**

Query: **prefix:MEMORY_SIZE?**

Response: prefix:MEMORY_SIZE size

Arguments:

prefix	Prefix A or B specifies the plug-in
size	1k, 2k, 5k, 10k, 20k, 50k for 7242A 1k, 2k, 5k, 10k, 20k, 50k, 100k, 200k for 7242B 1k, 2k, 5k, 10k, 20k, 50k, 100k, 200k, 500k, 1M for Option-L1

Examples:

A:MEMORY_SIZE 2K Sets memory size for plug-in A to 2K.

A:MEMORY_SIZE? Reports current memory size to be 50K.
A:MEMORY_SIZE 50K

Note: The memory size affects the time/point, the number of samples collected, and if in sequence mode, the number of segments allowable.

See Also: SEGMENTS, TIME_DIV, INTERLEAVED

NUM_ACQ_CHAN NACH

Purpose: This command selects the number of channels which will acquire data. Selection of 4 channels provides data on all channels with a maximum of 50k per channel single shot and 200k per channel RIS. Reducing the number of channels acquiring data will provide 100K points per channel single shot for channels 1 and 2, or 200k points for channel 1.

Command: **prefix:NUM_ACQ_CHAN num_channels**

Query: **prefix:num_acq_chan?**

Response: prefix:NUM_ACQ_CHAN num_channels

Arguments:

prefix	Prefix A or B specifies the plug-in	
num_channels	4	Sets the number of channels acquiring data to 4.
	2	Sets the number of channels acquiring data to 2.
	1	Sets the number of channels acquiring data to 1.

Examples:

A:NUM_ACQ_CHAN 2	Acquires data on channels 1 and 2
B:NUM_ACQ_CHAN? B:NUM_ACQ_CHAN 4	Reports the number of channels acquiring data is 4.

Notes: Channels not acquiring data can be used as trigger sources.

See Also: MEMORY_SIZE, SEGMENTS, SEQ_TRIGRATE

OFFSET OFST

Purpose: Sets the voltage offset used to position the signal within the input range. The range of offset values depends on the Volts/div as shown in the table below.

Command: **prefix:OFFSET value**

Query: **prefix:OFFSET?**

Response: prefix:OFFSET value

Arguments:

prefix — Prefix A or B specifies the plug-in, followed by the channel number

value — The following table specifies the valid ranges of offset values:

VOLTS/DIV	OFFSET RANGE
5 mV	±240 mV
10 mV	±240 mV
20 mV	±240 mV
50 mV	±600 mV
100 mV	±1.20 V
200 mV	±2.40 V
500 mV	±6 V
1 V	±10 V
1 V with VAR GAIN	±4 V

* Offset ranges are adjustable in 0.02 division increments

Examples:

A1:OFFSET 2E-02 V Sets offset to .02 Volts on plug-in A, channel 1

A2:OFFSET? Reports current offset for plug-in A, channel 2 to
A2:OFFSET 1.0 V be 1.0 Volts

Note: Offset can be adjusted for each channel.

See Also: VOLT_DIV

SAMPLE_CLOCK SCLK

Purpose: Sets the external sample clock used to synchronize the actual sampling of the plugin with the input signal. The allowed frequency range is DC to 1GHz and the nominal amplitude accepted is 0bD. When the sample rate is 1 GS/s, the clock frequency must be less than or equal to 1GHs. Otherwise the frequency must be less than or equal to 800 MHz.

Command: **prefix:SAMPLE_CLOCK state**

Query: **prefix:SAMPLE_CLOCK?**

Response: prefix:SAMPLE_CLOCK state

Arguments:

prefix	Prefix A or B specifies plug-in.	
state	INT	specifies the plug-in's internal clock
	EXT	specifies user-supplied clock

Examples:

A:SAMPLE_CLOCK EXT	Sets plug-in A sample clock to external.
A:SAMPLE_CLOCK? A:SAMPLE_CLOCK INT	Reports current clock type being used for plug-in A to be internal.

SEGMENTS SEGS

Purpose: Sets the number of segments to be acquired while in sequence trigger mode. The number of segments, the timebase, the number of channels acquiring data, and the memory size will determine the number of data points per segment. The maximum allowable segments will vary with memory size, number of channels acquiring data, and sequence trigger rate selected.

Command: **prefix:SEGMENTS segs**

Query: **prefix:SEGMENTS?**

Response: prefix:SEGMENTS segs

Arguments: prefix Prefix A or B specifies the plug-in.

Memory Size	Max Number of Segments with Max Trig Rate Standard	Option-L1	Max # of segments with Max Points Per Segment
1k	50	50	4000
2k	100	100	4000
5k	200	200	4000
10k	500	500	4000
20k	1000	1000	4000
50k	1000	2000	4000
>100k	1000	4000	4000

Examples:

A:SEGMENTS 25 Sets number of segments to be acquired for plug-in A to be 25.

A:SEGMENTS?
A:SEGMENTS 25 Reports current setting for segments to be acquired for plug-in A to be 25.

See Also: MEMORY_SIZE, TDIV, SEQ_TRIGRATE, NUM_ACQ_CHAN

SEQ_OPTION (7242B only) SOPT

Purpose: Selects the type of sequence mode acquisition to be either Sequential or Synchronous Averaging. Sequential sequence acquires multiple segments and displays them individually. Synchronous Averaging sequence accumulates multiple segments and averages them together.

Command: **prefix:SEQ_OPTION opt**

Query: **prefix:SEQ_OPTION?**

Response: prefix:SEQ_OPTION opt

Arguments:

plugin	Prefix A or B specifies the plug-in.	
opt	SEGMENT	Sets sequence trigger mode to Sequential.
	SYNC_AVG	Sets sequence trigger mode to Synchronous Averaging.

Examples:

A:SEQ_OPTION SEGMENT	Enables sequential sequence for plug-in A.
A:SEQ_OPTION? A:SEQ_OPTION SEGMENT	Reports SOPT to be sequential seqence for plug-in A.

Notes: When SEQ_OPTION is set for SYNC_AVG, the type of Synchronous Averaging that is performed depends on SYNC_AVG_OPT. If it is STANDARD, then Synchronous Averaging is performed on all channels. If set to ALTERNATE, then Alternate Synchronous Averaging is performed on Channel 1 only. See section on TIMEBASE status screen in the Operator's manual for a explanation of these sequence modes.

See Also: SYNC_AVG_OPT, SWEEPS

SEQ_TRIGRATE SQRT

Purpose:

Sets the method in which segments get acquired during sequence trigger mode. This selection will affect the sample rate, time per point, points per segment, and the trigger rate (the time from one trigger to the next).

Selection of TRIGRATE will limit the segment size and number of segments to provide maximum re-arm rate after each trigger.

Selection of POINTS will reduce the re-arm rate to provide greater record length and more segments. This is also referred to as "Packed Sequence Mode".

The trigger rate for each method is directly proportional to the record length.

Command: **prefix:SEQ_TRIGRATE method**

Query: **prefix:SEQ_TRIGRATE?**

Response: prefix:SEQ_TRIGRATE method

Arguments:

prefix	Prefix A or B	specifies plug-in.
method	TRIGRATE	specifies maximize trigger rate
	POINTS	specifies maximize points per segment

Examples:

A:SEQ_TRIGRATE POINTS sets plug-in A to maximize points per segment

A:SEQ_TRIGRATE? reports current sequence maximize method being used is trigger rate

A:SEQ_TRIGRATE TRIGRATE

See Also: TIME_DIV, SEGMENTS, MAX_MEMORY, NUM_ACQ_CHAN

SWEEPS (7242B only) SWPS

Purpose: Sets the number of sweeps to be accumulated in Synchronous Averaging or Alternate Synchronous Averaging sequence trigger mode.

Command: **prefix:SWEEPS sweeps**

Query: **prefix:SWEEPS?**

Response: **prefix:SWEEPS sweeps**

Arguments:
plugin Prefix A or B specifies the plug-in.
sweeps A number from 2...50000 in increments of 1

Examples:

A:SWEEPS 199 Sets the number of sweeps to be accumulated for plug-in A to be 199.

A:SWEEPS? Reports current setting for sweeps
A:SWEEPS 199 to be accumulated for plug- in A to be 199.

See Also: SEG_OPTION, SEQ_OPTION

SYNC_AVG_OPT (7242B only) SAOP

Purpose: Selects the type of Synchronous Averaging to perform.

Command: **prefix:SYNC_AVG_OPT** opt

Query: **prefix:SYNC_AVG_OPT?**

Response: prefix:SYNC_AVG_OPT opt

Arguments:

plugin	Prefix A or B specifies the plug-in.	
opt	STANDARD	Sets synchronous averaging mode to standard acquisition.
	ALTERNATE	Sets synchronous averaging mode to alternate. In this mode data is acquired on Channel 1 only and alternates between both trigger slopes.

Examples:

A:SYNC_AVG_OPT STANDARD	Sets Synchronous Averaging to standard acquisition.
A:SYNC_AVG_OPT? **A:SYNC_AVG_OPT STANDARD**	Reports the sequence averaging mode to be standard.

Note: Refer to the section on TIMEBASE status screen in the Operator's manual for an explanation of the difference between Synchronous Averaging and Alternate Synchronous Averaging.

See Also: SEQ_OPTION, ART_REJECT, SWEEPS

TIME_DIV TDIV

Purpose: Sets the time per division in a 1-2-5 sequence from 200 psec/div to 10000 Sec/div. This selection will affect the sampling rate, time per point, and points per division of the acquisition and the displayed trace.

If the timebase falls below 2ns/div, then RIS will automatically be enabled. If RIS is enabled and the timebase rises based on the following conditions, RIS will automatically be disabled:

Max Memory Selection	RIS Timebase Ranges	Roll Mode Ranges
1k	200ps/div to 250ns/div	.5s/div to 10Ks/div
2k	200ps/div to 10ns/div	.5s/div to 10Ks/div
5k	200ps/div to 20ns/div	.5s/div to 10Ks/div
10k	200ps/div to 50ns/div	.5s/div to 10Ks/div
20k	200ps/div to 100ns/div	.5s/div to 10Ks/div
50k	200ps/div to 200ns/div	.5s/div to 10Ks/div
100k	200ps/div to 500ns/div	.5s/div to 10Ks/div
200k	200ps/div to 1µs/div	1s/div to 10Ks/div
*500k	200ps/div to 2µs/div	2s/div to 10Ks/div
		*5s/div to 10Ks/div

* Available only when 7242B is purchased with memory Option-L1

Command: **prefix:TIME_DIV tdiv**

Query: **prefix:TIME_DIV?**

Response: prefix:TIME_DIV tdiv

Arguments:
prefix Prefix A or B specifies the plug-in.
tdiv Range of values 200 ps to 10000 s in a 1–2–5 sequence.

Example:
A:TIME_DIV 20 NS Sets time/div on plug-in A to 20 nsec/div

A:TIME_DIV? Reports the current time per division to be 20 nsec/div
A:TME_DIV 20e -09

Note: Allowable units are PS, NS, US, MS, AND S.

See Also: INTERLEAVED, SEGMENTS, MEMORY_SIZE, SEQ_TRIGRATE

TRGDLY_UNIT TDUN

Purpose: Selects the method used for setting the trigger delay. Trigger delay can be adjusted as a function of time or screen percentage, the trigger point can be positioned anywhere on the screen and will be maintained as the time-base changes. When adjusted in time, the position of the trigger point will vary as the timebase changes.

Command: **prefix:TRGDLY_UNIT unit**

Query: **prefix:TRGDLY_UNIT?**

Response: prefix:TRGDLY_UNIT unit

Arguments:

prefix — Prefix A or B specifies the plug-in

unit
- TIME — Trigger delay adjusted in time
- PERCENT — Trigger delay in screen percentage

Examples:

A:TRGDLY_UNIT PERCENT	Sets trigger delay to be adjusted in time
A:TRFDLY_UNIT?	Reports current trigger delay unit
A:TRGDLY_UNIT TIME	

Note: When locked timebases is selected, delay controls are locked.

See Also: TRIG_DELAY

TRIG_COUPLING TRCP

Purpose: Selects the method used to couple the trigger source to the input of the trigger circuit.

Command: **prefix:TRIG_COUPLING couple**

Query: **prefix:TRIG_COUPLING?**

Response: prefix:TRIG_COUPLING couple

Arguments:

prefix	Prefix A or B specifies the plug-in followed by 1, 2, or 3 to specify trigger source	
couple	AC	signals are capactively coupled
	LFREJ	signals are coupled via a capacitive high-pass filter network which attenuates frequencies below 50 KHz
	HFREJ	signals are dc coupled to the trig circuit and a low-pass filter network attenuates frequencies above 50 KHz
	DC	signals are DC coupled

Examples:

A2:TRIG_COUPLING HFREJ
A — Sets trigger coupling to HF REJ on plug-in trigger source, channel 2.

A3:TRIG_COUPLING?
A3:TRIG_COUPLING AC — Reports current trigger coupling for channel 3 of plug-in A to be capacitively coupled.

Note: A different coupling can be selected for each trigger source. If trigger source is line, the coupling is set to AC.

When locked triggers is selected, trigger coupling cannot be modified on slave plug-ins.

See Also: TRIG_SELECT

TRIG_DELAY TRDL

Purpose: Sets the degree of pre- or post-trigger delay when recording signals in the acquisition memories. Delay is specified in terms of time.

Command: **prefix:TRIG_DELAY time**

Query: **prefix:TRIG_DELAY?**

Response: **prefix:TRIG_DELAY time**

Arguments:

prefix	Prefix A or B specifies the plug-in
time	Pretrigger low limit in the range of -5 x TIME_DIV. Post-trigger delay in the range of 100,000 x time/div.

Examples:

A:TRIG_DELAY 5.0 s	Sets trigger delay for plug-in A to 5 sec.
A:TRIG_DELAY? A:TRIG_DELAY 6.0 s	Reports current trigger delay for plug-in A to be 6 seconds.

Note: When locked timebases is selected, delay controls are locked.

TRIG_LEVEL TRLV

Purpose: Sets the signal level required to generate a trigger. When the trigger level is increased, the trigger circuit will respond at a higher voltage level. If vertical sensitivity is adjusted such that the previously selected trigger level exceeds the range, the trigger level is automatically adjusted to fit the new range.

Command: **prefix:TRIG_LEVEL value**

Query: **prefix:TRIG_LEVEL?**

Response: prefix:TRIG_LEVEL value

Arguments: prefix Prefix A or B specifies the plug-in followed by 1, 2, EX, or EX/10 to specify trigger source

value Each trigger source can have the following range of values:

TRIG_SOURCE	RANGE	INCREMENT
CHANNEL 1	+/-5 div x Total gain	.02 x total gain
CHANNEL 2	+/-5 div x Total gain	.02 x total gain
LINE	Power line voltage range	
EXTERNAL	±2volts	.002
EXT/10	±20volts	.01

Examples:

A3:TRIG_LEVEL 1.002 V Set trigger level to 1.002 V for channel 3 of plug-in A

A1:TRIG_LEVEL?
A1:TRIG_LEVEL .050 V
Report current trigger level for plug-in A's channel 1 to be 50 mV.

Notes: Trigger level can be adjusted for each trigger source independently. Total gain is the product of VDIV times ATTENUATION.

When locked triggers is selected, trigger level cannot be modified on slave plug-ins.

See Also: VDIV, ATTENUATION

TRIG_PATTERN TRPA

Purpose: Defines a trigger pattern. Specifies the logic composition of the pattern sources (CH1, CH2, EX, EX10) and the conditions under which a trigger can occur.

Command: **prefix:TRIG_PATTERN CH1_state,CH2_state,Ext_source, Ext state, trig_condition**

Query: **prefix:TRIG_PATTERN?**

Response: prefix:TRIG_PATTERN CH1_state, CH2_state, Ext source, Ext state, trig_condition

Arguments: prefix Prefix A or B specifies the plug-in.

CH1_state, CH2_state, Ext_state can have the following values:

L	Low
H	High
X	Don't care

Ext_source can have the following values:

EX	External trigger source
EX10	External/10 trigger source

trig_condition can have the following values:

PR	Pattern present
AB	Pattern absent
EN	Pattern entered
EX	Pattern exited

PR and AB are valid only for State Qualified trigger. EN and EX are valid only for Pattern and Time/event Qualified triggers.

Examples:

A:TRIG_PATTERN H,L,X,PR Trigger when CH1=H, CH2=L, and EX=X are present

A:TRIG_PATTERN? Reports current trigger pattern
A:TRIG_PATTERN H,L,X,PR

TRIG_PATTERN (continued) TRPA

Notes:

State is not used for the current trigger source when the trigger type is State Qualified. A query during this time will specify UNDEF for the state.

This command can be used even if Smart Trigger mode is not activated.

When locked triggers is selected, trigger pattern cannot be modified on slave plug-ins.

See Also: TRIG_SELECT

TRIG_SELECT TRSE

Purpose: Selects the actual condition that will trigger the acquisition of waveforms. Depending on the trigger type, additional parameters must be specified. The additional parameters are grouped in pairs. The first one specifies the keyword to be modified and the second one gives the new value to be assigned. Pairs may be given in any order and may be restricted to those variables to be changed.

Command: **prefix:TRIG_SELECT trig_type[, keyword, value [..., keyword, value]]**

Query: **prefix:TRIG_SELECT?**

Response: prefix:TRIG_SELECT trig_type[,keyword, value[..., keyword, value]]

Arguments: prefix Prefix A or B specifies the plug-in

trig_type Possible values:

STD	Standard trigger
SNG	Single source trigger
PA	Pattern trigger
SQ	State qualified trigger
TEQ	Time/event qualified trigger

keywords

SR TRIGGER SOURCE:

Associated values:	
CH1	Channel 1
CH2	Channel 2
EX	External
EX10	External/10
LINE	Line

SR does not apply to Pattern triggers

Note: Sending the TRIG_SELECT command to a slave plug-in in locked trigger mode will make it the trigger master.

TRIG_SELECT (continued) TRSE

HT HOLD TYPE

Possible values:

TI	holdoff by time
EV	holdoff by events
PS	Pulse less than
PL	Pulse greater than
IS	Interval less than
IL	Interval greater than
BFTI	Trigger before time expires

HT does not apply to Standard triggers.
PS, PL, IS, IL only applies to single source and pattern triggers.
BFTI only applies to State and Time/event qualified triggers.

HV HOLD VALUE

30 nsec – 680 sec	TI, IS, IL
10 nsec – 680 sec	BFTI
1 nsec – 680 sec	PS and PL
0 – 15.0e06	EV

HV does not apply to Standard triggers.

FR FRAME

ODD	Odd frame
EVEN	Even frame
BOTH	Non-Interlaced

Valid only for TV triggers.

LN LINE

Ranges 10-2500 lines

Valid only for TV triggers.

SCRATE SCAN RATE

LRES	Low resolution (15-20 KHz)
MRES	Medium resolution (20-30 KHz)
HRES	High resolution (30-63 KHz)

Valid only for TV triggers.

TRIG_SELECT (continued) TRSE

Examples:

B:TRIG_SELECT STD,SR,CH2 Standard trigger on Channel 2

B:TRIG_SELECT SNG,SR,LINE,HT,PS,HV,2.0NSEC Single Source trigger on line with pulse less than 2nsec

B:TRIG_SELECT PA,HT,EV,HV,4 Pattern trigger on four events

B:TRIG_SELECT SQ,SR,CH1,HT,BFTI,HV,300.0 NS State Qualified trigger on Channel 1 if conditions are met within 300 nsec

B:TRIG_SELECT? Report current trigger conditions
B:TRIG_SELECT SQ,SR,CH1,HT,BFTI,HV,300.0e-09s

Notes:

Pattern, State Qualified, and Time/event Qualified triggers use the trigger pattern defined by the command TRIG_PATTERN.

See Also:

TRIG_PATTERN, HF_SYNC

TRIG_SLOPE TRSL

Purpose: Sets the signal edge used to activate the trigger circuit.

Command: **prefix:TRIG_SLOPE edge**

Query: **prefix:TRIG_SLOPE?**

Response: prefix:TRIG_SLOPE edge

Arguments:

prefix	Prefix A or B specifies the plug-in followed by 1, 2, EX, or EX/10 to specify trigger source	
edge	POS	Triggers on a positive slope
	NEG	Triggers on a negative slope

Examples:

A2:TRIG_SLOPE POS Sets slope to positive on plug-in A when trigger source is channel 2

A3:TRIG_SLOPE? Report current slope value for plug-in A's channel 3 trigger source
A3:TRIG_SLOPE POS

Notes: Slope is not valid when the trigger source is LINE. Slope can be set independently for each trigger source. If HF_SYNC is on, the trigger slope is set to positive and cannot be modified.

When locked triggers is selected, slope cannot be modified on slave plug-ins.

VOLT_DIV VDIV

Purpose: Sets the vertical sensitivity in VOLTS/DIV. Sensitivity ranges from 5mv to 2.5V/div. This value combines the variable gain and the fixed volts/div.

Command: **prefix:VOLT_DIV value**

Query: **prefix:VOLT_DIV?**

Response: prefix:VOLT_DIV value

Arguments:

prefix	Prefix A or B specifies the plug-in followed by the channel
value	Range is 5.0 mV to 2.5 V

Examples:

A2:VOLT_DIV 2 V	Set total gain to 2 volts for plugin A channel 2
A1:VOLT_DIV?	Report current total gain for plug-in A
A1:VOLT_DIV .005 V	channel 1 to be 5 mV

Notes: This value does not contain probe attenuation.
VDIV value can be adjusted for each channel.

See Also: ATTENUATION, TRIG_LEVEL, OFFSET

LeCroy 7234 Plug-in Remote Commands

Organization

Each command description begins a new page. A command's name (header) is printed in long and short form near the top of the page. Although the long form is used in the description, the short form can be used instead. Below the header are up to eight sections which describe it:

Purpose: explains a command's use
Command: contains a command's syntax
Query: contains the query syntax
Response: contains the syntax of the response to a query
Argument: defines a choice(s)
Example: includes the command being used
Note: reports additional considerations
See Also: cites other relevant commands

Command Execution

Execution of program messages depends on the instrument status. As a rule, commands and queries can be executed in either Local or Remote mode.

Before attempting to execute a command or query, the parser scans it to verify its correctness and that sufficient information is given to perform a requested action.

ART_REJECT AREJ

Purpose: To turn artifact rejection on or off while in Synchronous Averaging. When enabled, any waveform with an underflow or overflow value is not included in the average. If artifact reject is turned off, any overflows are set to the maximum possible value of the ADC and any underflows to the minimum value.

Command: **prefix:ART_REJECT state**

Query: **prefix:ART_REJECT?**

Response: prefix:ART_REJECT state

Arguments: plugin Prefix A or B specifies the plug-in.

state ON Turns artifact rejection on.

OFF Turns artifact rejection off.

Examples:

A:ART_REJECT ON Sets artifact rejection on.

A:ART_REJECT? Reports the state of artifact
A:ART_REJECT ON as on.

See Also: SEQ_OPTION, SYNC_AVG_OPT, SWEEPS

ATTENUATION ATTN

Purpose: Sets the multiplier to the total vertical gain. If attenuation-coded probes are used, the probe coding contact rings surrounding the CH1, CH2, CH3 and CH4 BNC connectors recognize the attenuation factors of the probes, and this value cannot be modified with this command. If no probe multiplier is detected, this command can be used to set the attenuation reflecting the attenuation for the probe being used. The current value can always be queried.

Command: **prefix:ATTENUATION atten**

Query: **prefix:ATTENUATION?**

Response: prefix:ATTENUATION atten

Arguments:
prefix Prefix A or B specifies the plug-in followed by the channel number- 1,2,3, or 4
atten range 1, 10, 100, 1000

Examples:

A2:ATTENUATION 10	Sets probe attenuation for channel 2 on plug-in A to 10x.
A1:ATTENUATION? A1:ATTENUATION 1	Reports current probe attenuation for channel 1 on plugin A to be 1x.

Notes: A different probe attenuation can be selected for each channel.

The locking of vertical channel controls can only occur when the probe attenuations are equal.

See Also: VOLT_DIV

BANDWIDTH_LIMIT BWL

Purpose: Reduces the bandwidth from 500 MHz to 95 MHz. Use it to reduce signal and system noise, or to prevent high-frequency aliasing for sample rates above 40 MS/s.

Command: **prefix:BANDWIDTH_LIMIT state**

Query: **prefix:BANDWIDTH_LIMIT?**

Response: prefix:BANDWIDTH_LIMIT state

Arguments:

prefix	Prefix A or B specifies the plug-in followed by the channel number -1,2,3, or 4
state	Choices are ON or OFF

Examples:

A1:BANDWIDTH_LIMIT ON	Turns on bandwidth limit on plug-in A, channel 1.
B2:BANDWIDTH_LIMIT? B2:BANDWIDTH_LIMIT OFF	Reports current setting for plug-in B, channel 2 to be off.

Note: Bandwidth limit can be adjusted for each channel.

COUPLING CPL

Purpose: Sets the vertical coupling used to couple a signal to the vertical amplifier input. In the AC position, signals are coupled capacitively, thus blocking the inputs signal's DC component and limiting the lower signal frequencies to greater than 10 Hz. The input impedance is 1MΩ. In the DC position, all signal frequency components are allowed to pass through, and the impedance is selectable as 1 MΩ or 50Ω. If the 50Ω input is selected, the signal will be automatically disconnected from the amplifier whenever the maximum dissipation is exceeded. If this condition exists, the input coupling is switched to GND. To clear the overload condition, remove the signal from the input and reselect the desired coupling.

Command: **prefix:COUPLING couple**

Query: **prefix:COUPLING?**

Response: prefix:COUPLING couple

Arguments:

prefix	Prefix A or B specifies the plug-in followed by the channel number-1,2,3, or 4	
couple	A1M	AC coupling
	D1M	DC coupling with impedance at 1 MΩ
	GND	Signal is set to ground
	D50	DC coupling with impedance at 50 Ω

Examples:

A1:COUPLING A1M	Turns on AC coupling on plug-in A, channel 1.
A2:COUPLING? A2:COUPLING D50	Reports current coupling value for plug-in A, channel 2, to be DC coupling with impedance at 50 Ω.

Note: Coupling can be modified for each channel.

ENHANCED_RES ERES

Purpose: Selects the amount of digital filtering on a signal. By limiting the system bandwidth, noise can be filtered and reduced. Thus, the effective resolution can actually be improved beyond the ADC's ideal performance. Selecting greater values of this parameter increasingly filters noise. Since the ADCs each have 8 bits, selecting the lowest value removes the filter.

Command: **prefix:ENHANCED_RES res**

Query: **prefix:ENHANCED_RES?**

Response: prefix:ENHANCED_RES res

Arguments:

prefix	Prefix A or B specifies the plug-in followed by the channel number-1,2,3, or 4
res	Range 8.0 ... 11.0 in increments of .5.

Examples:

A2:ENHANCED_RES 10.0	Sets enhanced resolution on plug-in A , Channel 2
A2:ENHANCED_RES? A2:ENHANCED_RES 9.5	Reports current enhanced resolution on plug-in A, Channel 2 to be 9.5.

Note: Enhanced resolution can be selected for each channel.

The enhanced resolution filter will not be applied while in Sequence Trigger Mode.

HF_SYNC HFSY

Purpose: Selects whether the trigger source rate is divided. Set HF Sync to ON to allow stable triggering for sources greater than 200 MHz. This control can be used only when SMART TRIGGER is not selected.

Command: **prefix:HF_SYNC state**

Query: **prefix:HF_SYNC?**

Response: prefix:HF_SYNC state

Arguments:

prefix		Prefix A or B specifies the plug-in followed by the Trigger source to be modified - 1, 2, 3
state	ON	Sets high frequency sync ON. Trigger rate will be divided .
	OFF	Sets high frequency sync OFF.

Examples:

A1:HF_SYNC ON	Enables high freq sync for plug-in A when the trigger source is channel 1.
A3:HF_SYNC? A3:HF_SYNC OFF	Reports HFSY for plug-in A is disabled when trigger source is channel 3

Note: HF_SYNC can be selected for each trigger source.

See Also: TRIG_SELECT

INTERLEAVED ILVD

Purpose: Enables or disables Random Interleaved Sampling (RIS). RIS can be enabled under the following conditions:

MAX MEMORY SELECTION	RIS TIMEBASE RANGES
1K	200 ps/div to 5 ns/div
2K	200 ps/div to 10 ns/div
5K	200 ps/div to 20 ns/div
10K	200 ps/div to 50 ns/div
20K	200 ps/div to 100 ns/div
50K	200 ps/div to 200 ns/div
100K	200 ps/div to 500 ns/div
200K	200 ps/div to 1µs/div
* 500K	200 ps/div to 2µs/div
* 1M	200 ps/div to 5µs/div

* Available only when 7234 is purchased with memory option L1.

Command: **prefix:INTERLEAVED state**

Query: **prefix:INTERLEAVED?**

Response: prefix:INTERLEAVED state

Arguments:

prefix	Prefix A or B specifies the plug-in	
state	RANDOM	creates a RIS record with non-uniform sampling intervals.
	INTERPOLATED	creates a RIS record with uniform sampling intervals
	OFF	Turn RIS off

Examples:

A:INTERLEAVED RANDOM Turns interleaved sampling ON for plug-in A.

A:INTERLEAVED OFF Turns interleaved sampling OFF if timebase is greater than 5nsec/div.

A:INTERLEAVED?
A:INTERLEAVED RANDOM

Reports current value for plug-in A is interleaved sampling RANDOM.

INTERLEAVED (continued) ILVD

Note: The interpolated sampling option uses a linear interpolation algorithm and the nearest neighboring samples to create RIS record with uniform sampling intervals. The random sampling option does not perform this interpolation.

See Also: TIME_DIV

MEMORY_SIZE MSIZ

Purpose: Sets the maximum number of sample points to represent each waveform. By adjusting the maximum memory size, you can tradeoff record length for update rate. Acquire 1k samples to achieve maximum waveform throughput. Obtaining 200k samples will provide a greater time window of the signal but require more time to process the additional points.

Command: **prefix:MEMORY_SIZE size**

Query: **prefix:MEMORY_SIZE?**

Response: prefix:MEMORY_SIZE size

Arguments:

prefix	Prefix A or B specifies the plug-in	
size	1K	Nominal memory size equals 1K
	2K	Nominal memory size equals 2K
	5K	Nominal memory size equals 5K
	10K	Nominal memory size equals 10K
	20K	Nominal memory size equals 20K
	50K	Nominal memory size equals 50K
	100K	Nominal memory size equals 100K
	200K	Nominal memory size equals 200K
	* 500K	Nominal memory size equals 500K
	* 1M	Nominal memory size equals 1M

* Available only when 7234 is purchased with memory option L1.

Examples:

A:MEMORY_SIZE 2K Sets memory size for plug-in A to 2K.

A:MEMORY_SIZE? Reports current memory size to be 50K.
A:MEMORY_SIZE 50K

Note: The memory size affects the time/point, the number of samples collected, and if in sequence mode, the number of allowable segments.

See Also: SEGMENTS, TIME_DIV, INTERLEAVED, SEQ_TRIGRATE

OFFSET OFST

Purpose: Sets the voltage offset used to position the signal within the input range. The range of offset values depends on the Volts/div as shown in the table below.

Command: **prefix:OFFSET value**

Query: **prefix:OFFSET?**

Response: prefix:OFFSET value

Arguments:

prefix — Prefix A or B specifies the plug-in, followed by the channel number - 1,2,3, or 4.

value — The following table specifies the valid ranges of offset values:

VOLTS/DIV	OFFSET RANGE
5 mV	±240 mV
10 mV	±240 mV
20 mV	±240 mV
50 mV	±600 mV
100 mV	±1.20 V
200 mV	±2.40 V
500 mV	±6 V
1 V without vargain	±10V
1 V with vargain	±4V

Examples: **A1:OFFSET 2E-02 V** Sets offset to .02 Volts on plug-in A, channel 1

A2:OFFSET? Reports current offset for plug-in A, channel 2 to
A2:OFFSET 1.0 V be 1.0 Volts

Note: Offset can be adjusted for each channel.

See Also: VOLT_DIV

NUM_ACQ_CHAN NACH

Purpose: This command selects the number of channels which will acquire data. Selection of 4 channels provides data on all channels with a maximum of 50K per channel single shot and 200K per channel RIS. Reducing the number of channels acquiring data will provide 100K points per channel single shot for channels 1 and 2, or 200K points for channel 1.

Command: **prefix:NUM_ACQ_CHAN num_channels**

Query: **prefix: num_acq_chan?**

Response: prefix:NUM_ACQ_CHAN num_channels

Arguments:

prefix	Prefix A or B specifies the plug-in	
num_channels	4	Sets the number of channels acquiring data to 4.
	2	Sets the number of channels acquiring data to 2.
	1	Sets the number of channels acquiring data to 1.

Examples:

A:NUM_ACQ_CHAN 2	Acquires data on channels 1 and 2
B:NUM_ACQ_CHAN? B:NUM_ACQ_CHAN 4	Reports the number of channels acquiring data is 4.

Notes: Channels not acquiring data can be used as trigger sources.

See Also: MEMORY_SIZE, SEGMENTS, SEQ_TRIGRATE

SAMPLE_CLOCK SCLK

Purpose: Sets the external sample clock used to synchronize the actual sampling of the plug-in with the input signal. The allowed frequency range is DC to 200Mhz and the nominal amplitude accepted is 0db.

Command: **prefix:SAMPLE_CLOCK state**

Query: **prefix:SAMPLE_CLOCK?**

Response: prefix:SAMPLE_CLOCK state

Arguments:

prefix	Prefix A or B specifies plug-in.	
state	INT	specifies the plug-in's internal clock
	EXT	specifies user-supplied clock

Examples: **A:SAMPLE_CLOCK EXT** Sets plug-in A sample clock to external.

A:SAMPLE_CLOCK?	Reports current clock type being used for
A:SAMPLE_CLOCK INT	plug-in A to be internal.

SEGMENTS SEGS

Purpose: Sets the number of segments to be acquired while in sequence trigger mode. The number of segments, the timebase, the number of channels acquiring data, and the memory size will determine the number of data points per segment. The maximum allowable segments will vary with memory size, number of channels acquiring data, and sequence trigger rate selected.

Command: **prefix:SEGMENTS segs**

Query: **prefix:SEGMENTS?**

Response: prefix:SEGMENTS segs

Arguments: prefix Prefix A or B specifies the plug-in.

When Maximizing Trigger Rate:

SEGS	MEMORY_SIZE	MAX ALLOWABLE # OF SEGMENTS	MAX ALLOWABLE # OF SEGMENTS WITH MEMORY OPTION L1
	1K	50	50
	2K	100	100
	5K	200	200
	10K	500	500
	20K 4 CHANNELS	500	1000
	2 CHANNELS	1000	1000
	1 CHANNEL	1000	1000
	50K 4 CHANNELS	500	2000
	2 CHANNELS	1000	2500
	1 CHANNEL	1000	2500
	100K 4 CHANNELS	500	2000
	2 CHANNELS	1000	4000
	1 CHANNEL	1000	5000
	200K 4 CHANNELS	500	2000
	2 CHANNELS	1000	4000
	1 CHANNEL	1000	5000
	*500K 4 CHANNELS	NOT AVAILABLE	4000
	2 CHANNELS	NOT AVAILABLE	5000
	1 CHANNEL	NOT AVAILABLE	5000
	*1M 4 CHANNELS	NOT AVAILABLE	4000
	2 CHANNELS	NOT AVAILABLE	5000
	1 CHANNEL	NOT AVAILABLE	5000

* Available only when 7234 is purchased with memory option L1

SEGMENTS (continued) SEGS

When maximizing points per segment:

segs All memory sizes range 1...5000 in increments of 1

Examples:

A:SEGMENTS 25 Sets number of segments to be acquired for plug-in A to be 25.

A:SEGMENTS? Reports current setting for segments to be acquired
A:SEGMENTS 25 for plug-in A to be 25.

See Also: MEMORY_SIZE, TDIV, SEQ_TRIGRATE, NUM_ACQ_CHAN

SEQ_OPTION SOPT

Purpose: Selects the type of sequence mode acquisition to be either Sequential or Synchronous Averaging. Sequential sequence acquires multiple segments and displays them individually. Synchronous Averaging sequence accumulates multiple segments and averages them together.

Command: **prefix:SEQ_OPTION opt**

Query: **prefix:SEQ_OPTION?**

Response: prefix:SEQ_OPTION opt

Arguments:

plugin	Prefix A or B specifies the plug-in.	
opt	SEGMENT	Sets sequence trigger mode to Sequential.
	SYNC_AVG	Sets sequence trigger mode to Synchronous Averaging.

Examples:

A:SEQ_OPTION SEGMENT Enables sequential sequence for plug-in A.

A:SEQ_OPTION? Reports SOPT to be sequential
A:SEQ_OPTION SEGMENT seqence for plug-in A.

Notes: When SEQ_OPTION is set for SYNC_AVG, the type of Synchronous Averaging that is performed depends on SYNC_AVG_OPT. If it is STANDARD, then Synchronous Averaging is performed on all channels. If set to ALTERNATE, then Alternate Synchronous Averaging is performed on Channel 1 only. See section on TIMEBASE status screen in the Operator's manual for a explanation of these sequence modes.

See Also: SYNC_AVG_OPT, SWEEPS

SEQ_TRIGRATE SQRT

Purpose: Sets the method in which segments get acquired during sequence trigger mode. This selection will affect the sample rate, time per point, points per segment, and the trigger rate (the time from one trigger to the next). Selection of maximize trigger rate will tradeoff the number of segments and segment size for maximum re-arm rate. Selection of maximize points per segment will tradeoff re-arm rate for a larger number of segments and segment size. This method processes each segment as it is acquired. The trigger rate for each method varies with the length of each segment, and the number of channels acquiring data.

Command: **prefix:SEQ_TRIGRATE method**

Query: **prefix:SEQ_TRIGRATE?**

Response: prefix:SEQ_TRIGRATE method

Arguments:

prefix	Prefix A or B specifies plug-in.
method	TRIGRATE specifies maximize trigger rate
	POINTS specifies maximize points per segment

Examples:

A:SEQ_TRIGRATE POINTS	sets plug-in A to maximize points per segment
A:SEQ_TRIGRATE?	reports current sequence maximize method being used is trigger rate
A:SEQ_TRIGRATE	TRIGRATE

See Also: TIME_DIV, SEGMENTS, MAX_MEMORY, NUM_ACQ_CHAN

SWEEPS SWPS

Purpose: Sets the number of sweeps to be accumulated in Synchronous Averaging or Alternate Synchronous Averaging sequence trigger mode.

Command: **prefix:SWEEPS sweeps**

Query: **prefix:SWEEPS?**

Response: **prefix:SWEEPS sweeps**

Arguments:
plugin Prefix A or B specifies the plug-in.
sweeps A number from 2...50000 in increments of 1

Examples:

A:SWEEPS 199 Sets the number of sweeps to be accumulated for plug-in A to be 199.

A:SWEEPS? Reports current setting for sweeps
A:SWEEPS 199 to be accumulated for plug- in A to be 199.

See Also: SEG_OPTION, SEQ_OPTION

SYNC_AVG_OPT SAOP

Purpose: Selects the type of Synchronous Averaging to perform.

Command: **prefix:SYNC_AVG_OPT** opt

Query: **prefix:SYNC_AVG_OPT?**

Response: prefix:SYNC_AVG_OPT opt

Arguments:

plugin	Prefix A or B specifies the plug-in.	
opt	STANDARD	Sets synchronous averaging mode to standard acquisition.
	ALTERNATE	Sets synchronous averaging mode to alternate. In this mode data is acquired on Channel 1 only and alternates between both trigger slopes.

Examples:

A:SYNC_AVG_OPT STANDARD	Sets Synchronous Averaging to standard acquisition.
A:SYNC_AVG_OPT? A:SYNC_AVG_OPT STANDARD	Reports the sequence averaging mode to be standard.

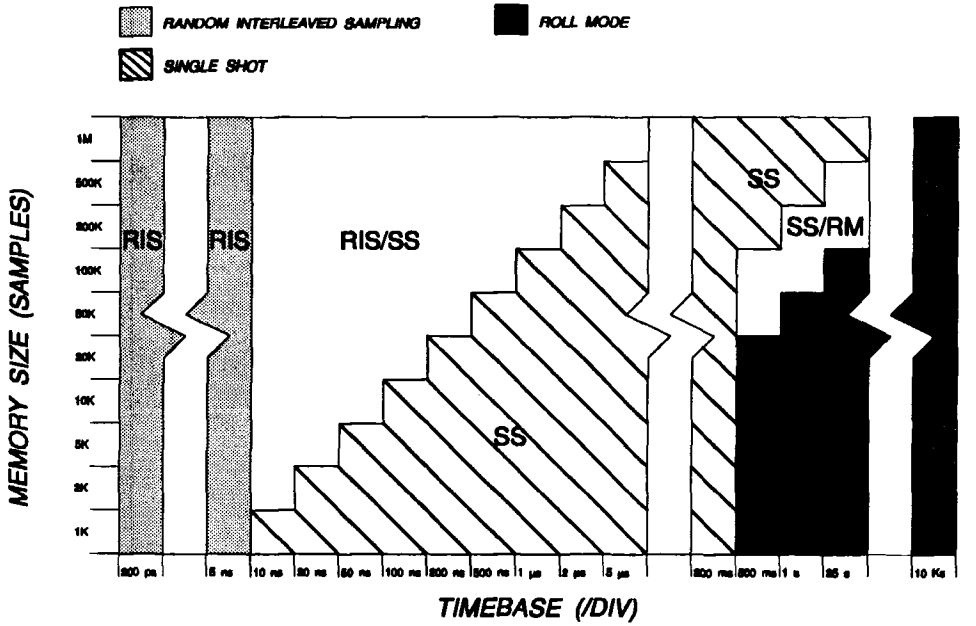
Note: Refer to the section on TIMEBASE status screen in the Operator's manual for an explanation of the difference between Synchronous Averaging and Alternate Synchronous Averaging.

See Also: SEQ_OPTION, ART_REJECT, SWEEPS

TIME_DIV TDIV

Purpose: Sets the time per division in a 1-2-5 sequence from 200 psec/div to 10000 Sec/div. This selection will affect the sampling rate, time per point, and points per division of the acquisition and the displayed trace.

If the timebase falls below 10 ns/div, then RIS will automatically be enabled. if RIS is enabled, RIS will be disable when no longer possible according to the table below. If the Timebase rises above 200ms/div Roll Mode will be enabled. if the timebase falls below 500ms/div Roll Mode will be disabled, except id the memory size is 200K then Roll Mode will be used at 1 s/div and above.



Command: **prefix:TIME_DIV tdiv**

Query: **prefix:TIME_DIV?**

Response: prefix:TIME_DIV tdiv

Arguments:

prefix	Prefix A or B specifies the plug-in.
tdiv	Range of values 200 ps to 10000 s in a 1–2–5 sequence.

TIME_DIV (continued) TDIV

Example:	**A:TIME_DIV 20 NS**	Sets time/div on plug-in A to 20 nsec/div
	A:TIME_DIV? A:TME_DIV 20e -09	Reports the current time per division to be 20 nsec/div
Note:	Allowable units are PS, NS, US, MS, AND S.	
See Also:	INTERLEAVED, SEGMENTS, MEMORY_SIZE, SEQ_TRIGRATE	

TRGDLY_UNIT TDUN

Purpose: Selects the method used for setting the trigger delay. Trigger delay can be adjusted as a function of time or screen percentage. When delay is adjusted in screen percentage, the trigger point can be positioned anywhere on the screen and will be maintained as the timebase changes. When adjusted in time, the position of the trigger point will vary as the timebase changes.

Command: **prefix:TRGDLY_UNIT unit**

Query: **prefix:TRGDLY_UNIT?**

Response: prefix:TRGDLY_UNIT unit

Arguments:

prefix	Prefix A or B specifies the plug-in	
unit	TIME	Trigger delay adjusted in time
	PERCENT	Trigger delay adjusted in screen percentage

Examples:

A:TRGDLY_UNIT PERCENT Sets the trigger delay to be adjusted in time

A:TRGDLY_UNIT? Reports current trigger delay unit

A:TRGDLY_UNIT TIME

Note: When locked timebases is selected, delay controls are locked.

See Also: TRIG_DELAY

TRIG_COUPLING TRCP

Purpose: Selects the method used to couple the trigger source to the input of the trigger circuit.

Command: **prefix:TRIG_COUPLING couple**

Query: **prefix:TRIG_COUPLING?**

Response: prefix:TRIG_COUPLING couple

Arguments:

prefix	Prefix A or B specifies the plug-in followed by 1, 2, or 3 to specify trigger source	
couple	AC	signals are capactively coupled
	LFREJ	signals are coupled via a capacitive high-pass filter network which attenuates frequencies below 50 KHz
	HFREJ	signals are dc coupled to the trig circuit and a low-pass filter network attenuates frequencies above 50 KHz
	DC	signals are DC coupled

Examples:

A2:TRIG_COUPLING HFREJ A	Sets trigger coupling to HF REJ on plug-in trigger source, channel 2.
A3:TRIG_COUPLING? A3:TRIG_COUPLING AC	Reports current trigger coupling for channel 3 of plug-in A to be capacitively coupled.

Note: A different coupling can be selected for each trigger source. If trigger source is line, the coupling is set to AC.

When locked triggers is selected, trigger coupling cannot be modified on slave plug-ins.

See Also: TRIG_SELECT

TRIG_DELAY TRDL

Purpose: Sets the degree of pre- or post-trigger delay when recording signals in the acquisition memories. Delay is specified in terms of time or screen percentage.

Command: **prefix:TRIG_DELAY value**

Query: **prefix:TRIG_DELAY?**

Response: **prefix:TRIG_DELAY value**

Arguments:

prefix	Prefix A or B specifies the plug-in
value	Pretrigger low limit in the range of -5 x TIME_DIV. Post-trigger delay in the range of 100,000 x time/div. Trigger delay percent in the range of 0% to 100%

Examples:

A:TRIG_DELAY 5.0 s	Sets trigger delay for plug-in A to 5 sec.
A:TRIG_DEALY 50%	Sets trigger point to center of screen.
A:TRIG_DELAY?	Reports current trigger delay for plug-in A to be
A:TRIG_DELAY 6.0 s6	seconds.

Note: When locked timebases is selected, delay controls are locked.

See Also: TRGDLY_UNIT

TRIG_LEVEL TRLV

Purpose: Sets the signal level required to generate a trigger. When the trigger level is increased, the trigger circuit will respond at a higher voltage level. If vertical sensitivity is adjusted such that the previously selected trigger level exceeds the range, the trigger level is automatically adjusted to fit the new range.

Command: **prefix:TRIG_LEVEL value**

Query: **prefix:TRIG_LEVEL?**

Response: prefix:TRIG_LEVEL value

Arguments:

prefix Prefix A or B specifies the plug-in followed by 1, 2, or 3 to specify trigger source

value Each trigger source can have the following range of values:

TRIG_SOURCE	RANGE	INCREMENT
CHANNEL 1	+/-5 div x Total gain	.02 x total gain
CHANNEL 2	+/-5 div x Total gain	.02 x total gain
CHANNEL 3	+/-5 div x Total gain	.02 x total gain

Examples:

A3:TRIG_LEVEL 1.002 V Set trigger level to 1.002 V for channel 3 of plug-in A

A1:TRIG_LEVEL?
A1:TRIG_LEVEL .050 V

Report current trigger level for plug-in A's channel 1 to be 50 mV.

Notes: Trigger level can be adjusted for each trigger source independently. Total gain is the product of VDIV times ATTENUATION.

When locked triggers is selected, trigger level cannot be modified on slave plug-ins.

See Also: VDIV, ATTENUATION

TRIG_PATTERN TRPA

Purpose: Defines a trigger pattern. Specifies the logic composition of the pattern sources (CH1, CH2, CH3) and the conditions under which a trigger can occur.

Command: **prefix:TRIG_PATTERN CH1_state,CH2_state,CH3_state, trig_condition**

Query: **prefix:TRIG_PATTERN?**

Response: prefix:TRIG_PATTERN CH1_state, CH2_state, CH3_state, trig_condition

Arguments: prefix Prefix A or B specifies the plug-in.

CH1_state, CH2_state, CH3_state can have the following values:

L	Low
H	High
X	Don't care

trig_condition can have the following values:

PR	Pattern present
AB	Pattern absent
EN	Pattern entered
EX	Pattern exited

PR and AB are valid only for State Qualified trigger. EN and EX are valid only for Pattern and Time/event Qualified triggers.

Examples:

A:TRIG_PATTERN H,L,X,PR Trigger when CH1=H, CH2=L, and CH3=X are present

A:TRIG_PATTERN? Reports current trigger pattern
A:TRIG_PATTERN H,L,X,PR

TRIG_PATTERN (continued) TRPA

Notes:

State is not used for the current trigger source when the trigger type is State Qualified. A query during this time will specify UNDEF for the state.

This command can be used even if Smart Trigger mode is not activated.

When locked triggers is selected, trigger pattern cannot be modified on slave plug-ins.

See Also: TRIG_SELECT

TRIG_SELECT TRSE

Purpose: Selects the actual condition that will trigger the acquisition of waveforms. Depending on the trigger type, additional parameters must be specified. The additional parameters are grouped in pairs. The first one specifies the keyword to be modified and the second one gives the new value to be assigned. Pairs may be given in any order and may be restricted to those variables to be changed.

Command: **prefix:TRIG_SELECT trig_type[, keyword, value [..., keyword, value]]**

Query: **prefix:TRIG_SELECT?**

Response: prefix:TRIG_SELECT trig_type[,keyword, value[..., keyword, value]]

Arguments: prefix Prefix A or B specifies the plug-in

trig_type Possible values:

STD	Standard trigger
SNG	Single source trigger
PA	Pattern trigger
SQ	State qualified trigger
TEQ	Time/event qualified trigger

keywords

SR TRIGGER SOURCE:

Associated values:

CH1	Channel 1
CH2	Channel 2
CH3	Channel 3
LINE	Line

SR does not apply to Pattern triggers

Note: Sending the TRIG_SELECT command to a slave plug-in in locked trigger mode will make it the trigger master.

TRIG_SELECT (continued) TRSE

HT HOLD TYPE

Possible values:

TI	holdoff by time
EV	holdoff by events
PS	Pulse less than
PL	Pulse greater than
IS	Interval less than
IL	Interval greater than
BFTI	Trigger before time expires

HT does not apply to Standard triggers.
PS, PL, IS, IL only applies to single source and pattern triggers.
BFTI only applies to State and Time/event qualified triggers.

HV HOLD VALUE

30 nsec – 680 sec	TI, IS, IL
10 nsec – 680 sec	BFTI
1 nsec – 680 sec	PS and PL
0 – 15.0e06	EV

HV does not apply to Standard triggers.

Examples:

B:TRIG_SELECT STD,SR,CH2 Standard trigger on Channel 2

B:TRIG_SELECT SNG,SR,LINE,HT,PS,HV,2.0NSEC Single Source trigger on line with pulse less than 2nsec

B:TRIG_SELECT PA,HT,EV,HV,4 Pattern trigger on four events

B:TRIG_SELECT SQ,SR,CH1,HT,BFTI,HV,300.0 NS State Qualified trigger on Channel 1 if conditions are met within 300 nsec

B:TRIG_SELECT? Report current trigger conditions
B:TRIG_SELECT SQ,SR,CH1,HT,BFTI,HV,300.0e-09s

Notes:

Pattern, State Qualified, and Time/event Qualified triggers use the trigger pattern defined by the command TRIG_PATTERN.

See Also:

TRIG_PATTERN, HF_SYNC

TRIG_SLOPE TRSL

Purpose: Sets the signal edge used to activate the trigger circuit.

Command: **prefix:TRIG_SLOPE edge**

Query: **prefix:TRIG_SLOPE?**

Response: prefix:TRIG_SLOPE edge

Arguments:

prefix	Prefix A or B specifies the plug-in followed by 1, 2, or 3 to specify trigger source	
edge	POS	Triggers on a positive slope
	NEG	Triggers on a negative slope

Examples:

A2:TRIG_SLOPE POS	Sets slope to positive on plug-in A when trigger source is channel 2
A3:TRIG_SLOPE? A3:TRIG_SLOPE POS	Report current slope value for plug-in A's channel 3 trigger source

Notes: Slope is not valid when the trigger source is LINE. Slope can be set independently for each trigger source. If HF_SYNC is on, the trigger slope is set to positive and cannot be modified.

When locked triggers is selected, slope cannot be modified on slave plug-ins.

VOLT_DIV VDIV

Purpose: Sets the vertical sensitivity in VOLTS/DIV. Sensitivity ranges from 5mv to 2.5V/div. This value combines the variable gain and the fixed volts/div.

Command: **prefix:VOLT_DIV value**

Query: **prefix:VOLT_DIV?**

Response: prefix:VOLT_DIV value

Arguments:

prefix	Prefix A or B specifies the plug-in followed by the channel 1,2,3 or 4
value	Range is 5.0 mV to 2.5 V

Examples:

A2:VOLT_DIV 2 V	Set total gain to 2 volts for plugin A channel 2
A1:VOLT_DIV?	Report current total gain for plug-in A
A1:VOLT_DIV .005 V	channel 1 to be 5 mV

Notes: This value does not contain probe attenuation.
VDIV value can be adjusted for each channel.

See Also: ATTENUATION, TRIG_LEVEL, OFFSET

Section 7: Internal Command Language

Index of ICL Statements and Functions

Introduction

This section describes the Internal Command Language (ICL), a programming language for customizing the 7200A to meet your specific applications. Using ICL, you can write programs to collect and examine data and make decisions about the result.

In many ways ICL is similar to Basic, C, or Pascal. A program consists of variables, subroutines, input and output statements, and statements which control the program's order of execution. In addition, an ICL program may contain any remote command or query described in sections 5 and 6 of this manual.

> *NOTE: When an ICL Program is running, remote commands are locked out. Therefore, bit 7 of the main status byte should always be checked (using a serial poll) before sending remote commands to ensure proper remote operation.*

This section is divided into subsections which describe:

Writing a Program
Example Programs
Elements of a Program
Statements
Built-in Functions

Writing a Program

The 7200A can learn a sequence of front-panel operations, as described in the 7200A Precision Digital Oscilloscope Operator's Manual. Using this facility you can create an ICL program consisting of a series of remote commands.

Learning a program is a good way to get started with a custom application. After the sequence of commands is learned, you can transfer it to a DOS formatted 3.5" floppy disk and use an IBM/PC or compatible to modify it or add flow-of-control statements.

The objective is to create a file containing the source text of your program. You can use any text editor to create the file, which should have the extension "SRC". For example, to write a program to write "Hello world." on the display, use any text editor to create a file called HELLO.SRC which contains the following four lines:

```
; This program prints on the screen.

print display 'Hello world.'
end
```

Now insert the floppy in the 7200A's disk drive and enter the Program Setup screen by pressing MODIFY and then "Learn Program" or "Run Program". Set the disk parameter to FLOPPY and move the box to the INPUT FILE entry. You should see your program HELLO among the choices. Select it and press the "Recall" key to read it. The 7200A will automatically compile the program and if any errors occur it will display a message, otherwise the text of the program will appear in the upper half of the screen.

Alternately, you can use the PC program called COMPILE, which is supplied with the 7200A. To compile the program before trying to load the program into the 7200A, Type

```
C>COMPILE HELLO
```

to create the file HELLO.APD. If you have been working on a hard disk drive, copy these two files onto a 3.5" floppy. With the floppy in drive B, type the following DOS command:

```
C>COPY HELLO.* B:
```

Press Return to display the Main Screen. Next press "Run Program" to execute the program. The message "Hello world." will appear in the upper half of the screen, indicating that your program has run successfully.

Example Programs

This section introduces the features of ICL using some example programs. The first example is a short program, just to familiarize you with the appearance of ICL programs. The second example more fully illustrates the features of ICL, and performs some operations which are common to many actual applications.

The line numbers given in the right-hand margin are used during the discussion of the programs, and are not actually included in the programs.

Example 1

The first example is straightforward. The comments within the program describe what each group of lines does. After the example is a detailed discussion of each group. As you read through the example, keep the following rules in mind:

- Lines beginning with a semicolon (;) are comments.
- Lines beginning with upper case letters are remote commands, as described in sections 5 and 6 of this manual.

- Lines beginning with lower case letters are ICL statements, and are used to control the program.

```
; Set trace 1 to be channel A1.                                    1
  T1:DEF EQN,"A1",MAXPTS,50000                                     2
                                                                   3
; Turn trace 1 on and the others off.                              4
  T1:TRA ON                                                        5
  for i = 2 to 8                                                   6
    'T' | i | ':TRA OFF'                                           7
  endloop                                                          8
                                                                   9
; Select a single grid                                            10
  GRID SINGLE, SINGLE                                              11
                                                                   12
; Perform autosetup and wait for it to finish.                     13
  ASET                                                             14
  wait for 20 seconds                                              15
                                                                   16
; Get the V/DIV setting for channel A1.                            17
  value = query('A1:VDIV?')                                        18
                                                                   19
; Print a message about the size of the signal.                    20
  if token(value, 1) <= 0.1                                        21
    print display 'Small Signal'                                   22
  else                                                             23
    print display 'Large Signal'                                   24
  endif                                                            25
end                                                                26
```

Lines	Description
1	This is a comment which describes line 2. In these example programs, indention (space at the beginning of the line) is used to help organize the program. The lines described by a comment are indented under it, and are followed by a blank line.
2	This is a remote command which is used to define the equation for trace 1. . For more information about this command, see section 5 of this manual, "Mainframe Remote Commands".
4-8	These lines adjust the traces so that trace 1 is on, and the others are off. Line 5 turns trace 1 on using the "TRACE" command, described in section 5. Lines 6 through 8 are a loop which turns traces 2 through 8 off.

The loop is introduced in line 6 with the *for* statement. This defines a variable, "i", which takes on the values 2, 3, 4, ... 8 on successive iterations of the loop. Line 8 terminates the loop. The lines within the loop (in this case line 7) are executed once for each different value of the variable "i".

Line 7 is a command which is constructed using the string concatenate (join) operator. The following items are joined to produce the command: the string 'T', the value of the variable "i", and the string ':TRA OFF'. For example, the first time through the loop, "i" has the value 2. The strings 'T', '2', and ':TRA OFF', are joined to produce the command 'T2:TRA OFF'. This is another use of the "TRACE" command which was used in line 5.

10-11 These lines use the "GRID" command to select a single grid.

13-15 These start an "AUTO_SETUP" and wait for it to finish. This brings up an important point: most commands begin actions, but don't wait for them to finish.

Sometimes, it isn't necessary to wait for a command to finish, as in the "GRID" command in line 11. In that case, it really doesn't matter when the change occurs.

In other cases, it is very important to wait for a command to finish. In particular, if the program needs a value which results from the command in question, you must wait for it to finish. This is done with the wait statement in line 15. This simply delays execution of the next operation in the program for 20 seconds, which should be enough time for the auto setup to complete.

Note that there is a better way to wait for operations to complete than to delay for a fixed amount of time. The *status* statements, along with a variation of the *wait* statement are used for this purpose. This method is discussed in the second example program.

17-18 These lines read the current value of the VOLTS/DIVISION setting using the query function, described on p. 7-80. The expression in parentheses, 'A1:VDIV?' is a variation of the "VOLT_DIV" command which, instead of setting the VOLTS/DIVISION for channel A1, returns the current value. This type of command is referred to as a query.

The VOLT_DIV query returns a string such as 'A1:VOLT_DIV .005 V'. The query function returns this string, less the heading 'A1:VOLT_DIV'. Therefore, the result could be '.005 V'.

20-25

These lines examine the VOLTS/DIVISION setting and print a message based on the value. The *token* function extracts the first token from the string returned by *query* in line 18. The first token is the number, such as '.005'.

Line 21 contains an *if* statement which tests the result. If it is less than or equal to 0.1, the *print* statement in line 22 is executed. Otherwise, the *print* statement in line 24 is executed.

Line 23 contains an *else* statement, which serves to separate the alternative choices of the *if* . Line 25 contains an *endif* statement, which indicates the end of the alternatives.

This example uses only one statement in each alternative, but as many statements as needed could have been used instead.

Lines 22 and 24 contain *print* statements, which are described on page 7-44. The keyword *display* indicates that the output is to be written at the top of the 7200A's screen. In both lines, the expressions following *display* are strings. If necessary, they could have joined the contents of variables and strings to produce more complex results.

26 This statement indicates the end of the program.

Example 2

The second example is more involved than the first. The first thing you might note is that it is divided into sections. Lines 1 to 27 are the main program. Lines 29 to 76 are the procedure "initialize". Lines 78 to 94 are the menu "set_up", etc.

There are several reasons for this type of structure. When a program grows beyond a certain size, it becomes difficult to understand. By dividing the program into procedures, you can isolate the parts which perform distinct functions. Each part is then smaller and easier to understand.

Making a procedure also provides the opportunity to call it from several different places. This results in a smaller program than would be the case if the procedure's statements were included everywhere they were needed.

Menus are used to build setup screens, like those used elsewhere in the 7200A. They enable you to display and modify program variables in a familiar way. The statements used in a menu are different than those used in the main program or a procedure. Therefore, the structuring of the program into menus is necessary if you want to use setup screens.

```
;*************************************************                1
; This program samples an input, counting parameter             2
; values which are above, below, and between two                3
; selected values. After all waveforms are analyzed,            4
; a report is generated.                                        5
                                                                6
  ; Define the choice of waveform parameters.                    7
    list parameters is (pkpk ampl min max)                      8
                                                                9
  ; Initialize variables.                                       10
    call initialize                                             11
                                                                12
  ; This loop examines "count" waveforms.                       13
    for i = 1 to count                                          14
      ; Acquire the next waveform.                              15
        call acquire                                            16
                                                                17
       ; Get the parameter value and update totals.             18
        call update                                             19
    endloop                                                     20
                                                                21
  ; Select normal trigger mode.                                 22
    TRMD NORM                                                   23
                                                                24
   ; Report the results.                                       25
    call report                                                 26
end                                                             27
                                                                28
;*************************************************              29
; This procedure sets initial values of variables               30
; and puts the 7200A into a known state.                        31
                                                                32
procedure initialize                                            33
  ; Define trace 1 to be channel A1 with no processing.         34
    T1:DEF EQN,"A1",MAXPTS,50000                                35
                                                                36
  ; Turn trace 1 on and the others off.                         37
    T1:TRA ON                                                   38
    list traces_off is (T2 T3 T4 T5 T6 T7 T8)                   39
    for Trace in traces_off                                     40
```

```
      Trace:TRA OFF                                                    41
    endloop                                                            42
                                                                       43
  ; Select single grid.                                              44
    GRID SINGLE , SINGLE                                               45
                                                                       46
  ; Select normal trigger mode. This will provide the                47
  ; user with a live display during the menu "set_up"                48
  ; to allow adjustment of acquisition parameters.                    49
    TRMD NORM                                                          50
                                                                       51
  ; Set default values of count, high, and low if                     52
  ; count is currently undefined.                                     53
    if type(count) = 2                                                 54
      count = 100                                                      55
      high = 10.0                                                      56
      low = -10.0                                                      57
    endif                                                              58
                                                                       59
  ; Call the menu "set_up" to set the program choices.                60
    call set_up                                                        61
                                                                       62
  ; Clear the value counters.                                         63
    above = 0                                                          64
    inside = 0                                                         65
    below = 0                                                          66
    invalid = 0                                                        67
                                                                       68
  ; Stop any acquisitions currently in progress.                      69
    STOP                                                               70
                                                                       71
  ; Initialize status for the wait statement in "acquire".            72
      status disable                                                   73
      status clear                                                     74
      status enable 'PROCESSING DONE 1'                                75
end                                                                    76
                                                                       77
```

```
;*************************************************
; This menu allows the operator to select the waveform
; parameter to be measured, the number of waveforms
; to be measured, and the upper and lower bounds
; for the parameter value

menu set_up
  display 'Sample Program Setup'
  display
  var parm list parameters          display 'Parameter   '
  var count integer 1 to 5000       display 'Count       '
  var high real -10 to 10 step 0.001 display 'Upper Limit '
  var low real -10 to 10 step 0.001  display 'Lower Limit  '
  display
  display 'Apply signal to  A1 and adjust.'
  display 'Press RETURN to continue.'
end

;*************************************************
; This procedure acquires a waveform.

procedure acquire
  ; Clear any previous status indicating processing is
  ; done for trace 1.
    status clear 'PROCESSING DONE 1'

  ; Arm the input channel to initiate acquisition.
    ARM

  ; Wait for processing to be completed on trace 1.
     wait for any status or 30 seconds

  ; Check to see if a timeout occurred.
    if cond('TIMEOUT')
      ; Print an error message on the screen.
        print display 'Timeout occurred waiting for input.'

      ; Stop the program immediately.
        PRMO OFF
    endif
end
```

```
;*************************************************
; This procedure reads the value of the selected
; parameter and updates the appropriate counter.

procedure update
  ; Get the value of the waveform parameter. An
  ; example of a string returned by this query is:
  ;          AMPL,1.5 V,OK
    value = query('T1:PAVA?' || parm)

  if search(value, 'OK') != -1
    ; The parameter was OK.

    ; Extract the value from the result.
      value = token(value, 3)

    ; Update the appropriate counter.
      if value > high
        above = above + 1
      elseif value < low
        below = below + 1
      else
        inside = inside + 1
      endif
  else
    ; Update the "invalid" count.
      invalid = invalid + 1
  endif
end
```

```
;*************************************************                    150
; This menu displays the result of the waveform                      151
; measurements in a tabular form.                                    152
                                                                     153
menu report                                                          154
  display '    Sample Program Report'                                155
  display                                                            156
  display '# of waveforms   value of' || parm                        157
  display                                                            158
  display format('    %4d       above %g', above, high)              159
  display format('    %4d       %g to %g', inside, low, high)        160
  display format('    %4d       below %g', below, low)               161
  display format('    %4d       invalid', invalid)                   162
  display                                                            163
  display 'Press HARDCOPY to print this report.'                     164
  display 'Press RETURN to continue.'                                165
end                                                                  166
```

Lines	Description
1-5	These lines are comments which describe the purpose of the program.
7-8	The *list* statement is used to define the choices of waveform parameters which can be measured by the program. It defines a list, "parameters", which is a collection of items which can be identifiers, strings, or numbers. The name "parameters" can be used to denote the entire list, as in the *var* statement in line 87. In this case, the list contains the entries "pkpk", "ampl", "min", and "max". The menu "set_up" sets the variable "parm" to one of these for use in the query function in line 128.
10-11	These lines call the procedure "initialize" to set up variables and conditions in the 7200A.
13-20	This is the main loop in the program. It calls the procedures "acquire" and "update" the number of times specified in the variable "count". This variable is selected by the user in the menu "set_up".
22-23	This sets the trigger to NORMAL, which provides a live display of data when the menu "report" is called.
25-26	The menu "report" is called to display the results.

27 This *end* statement terminates the main program. Note that the main program is always at the beginning of the program, before any menus or procedures. Menus and procedures may appear in any convenient order after the main program.

29-33 This is the beginning of the procedure "initialize".

34-35 This is a use of the "DEFINE" command, which is used to define the equation for trace 1. The prefix "T1" indicates that trace 1 is being defined. The "EQN" keyword indicates that the equation for the trace follows the next comma. The equation itself is always enclosed in double quotes. The "MAXPTS" keyword indicates that the maximum number of points (horizontal length of the waveform) is given after the next comma. For more information about this command, see section 5 of this manual, "Mainframe Remote Commands" .

37-42 These lines adjust the traces so that trace 1 is on, and the others are off. Line 38 turns trace 1 on using the "TRACE" command, described in section 5. Line 39 defines a list of traces, similar to the list of parameters defined in line 8. Lines 40 through 42 are a loop which turns traces 2 through 8 off.

The loop is introduced in line 40 with the *for* statement. This defines a variable, "Trace", which takes on the values of the list "traces_off" ("T2", "T3", ... "T8") on successive iterations of the loop. Line 42 terminates the loop. The line within the loop (41) is executed once for each different value of the variable "Trace".

Line 41 is a command which begins with a variable, "Trace". Since it is a command, it must start with an upper case letter. For this reason, the first letter of the variable "Trace" is capitalized. Normally, character case does not matter in identifiers. This is the only exception.

When a variable or expression appears in a command, it is replaced by its value. This produces the commands "T2:TRA OFF", "T3:TRA OFF", etc.

This loop performs the same function as the loop in lines 4-8 of the first example program. However, that loop uses a different technique for constructing the "TRACE" command.

44-45 These lines use the "GRID" command to select a single grid.

47-48 These lines set the trigger to NORMAL. During the menu "set_up", which is called in line 61, the user will be given an opportunity to connect the input to channel A1 and adjust the controls to give a good view of the waveform. The live display provided by NORMAL triggering will allow the user to see the effects of any changes made. Also, we will be sure that the trigger parameters

are set correctly so that in the procedure "acquire", data will be acquired after the "ARM" command.

52-58 These lines provide default values for the numeric variables set by the menu "set_up". Lines 88-90 in that menu indicate how these variables may be adjusted by the user. However, if they are undefined when the menu is entered, they will be assigned the lowest value in the given range, which is not a particularly good choice in this case.

Line 54 begins an *if* statement which determines if a value has ever been stored in the variable "count". This is done using the *type* function, which is described on p. 7-92. The value 2 is returned if the argument is an undefined variable. Line 58 contains an *endif* statement which determines the end of the statements executed only if "count" is undefined.

Lines 55 to 57 set the values of the variables "count", "high", and "low". In addition to setting their values, these lines also indicate that the words "count", "high", and "low" are used by this program to represent variables. These words may be used anywhere in the program, and will always refer to the same variables.

60-61 These lines call the menu "set_up". A menu defines a setup screen, as used throughout the 7200A. While the menu is active, the softkeys and the knobs below the center of the screen are assigned new meanings as defined by the menu. When you are done setting values and want to continue with the rest of the program, press the "Return" ~~ppppp~~ or "Cancel Changes" key for that purpose if necessary. See p. 7-91 for more information.

63-67 These lines set to zero the variables used to count different values of the waveform parameter selected in the menu "set_up".

69-70 These lines stop any acquisition which may be in progress and select single trigger. The procedure "acquire" will use the "ARM" command to initiate an acquisition. Before doing so, the system must be in a known state.

72-75 These lines prepare the conditions for the *wait* statement on line 108 in the procedure "acquire". The purpose of that *wait* is to delay execution until the next waveform is acquired and processed for display in trace 1.

In the first example program, the wait statement in line 15 is used to wait for an action to complete by simply waiting for a fixed time. This tends to make the program slow because you have to make the delay long enough to work always. It is much better to wait until the event actually happens.

The 7200A provides status information as described in Section 4 of this manuals. This information is used by the *status* and *wait* statements which are described on pages 7-49 and 7-53.

Line 73 disables all status conditions. This is used prior to the *status* enable statement in line 75 to make sure that only one status condition is enabled.

Line 74 clears all pending status conditions. This line and the preceding one are not strictly necessary, since these operations are performed automatically when the program is started. They are included for illustration only.

Line 75 enables the status condition called "PROCESSING DONE 1". This is one of many different conditions supported by the *status* statement. Once a status condition is enabled, a *wait* statement may be used to delay execution until the associated event happens.

Note that not all status information described in section 4 of this manual is supported by the status statement. The information which describes error conditions is used by ICL to help control the program.

76 This line indicates the end of the procedure "initialize".

78-84 This is the beginning of the menu "set_up". The statements which appear in a menu serve one of two purposes. They may place an entry in the main part of the menu (where the grids are normally located) or next to a key.

In the main part of the menu, each entry occupies a line, begging at the top of the menu and working downwards. If no lines are used, the full screen is available for the display of traces. If there are 1 to 13 lines in the menu, the menu occupies the lower half of the screen, and the upper half of the screen is available for traces. If there are more than 13 lines, the menu occupies the entire screen.

The power operator, ^, raises one number to the power of another number. 5 ^ 3 is the same as 5 * 5 * 5, or 125. If the first operand is zero, the second must be greater than zero. If the first operand is negative, the second must be an integer.

The *mod* operator returns the remainder which is left after dividing one number by another. 10 mod 3 is 1, since 10/3 is 3 with a remainder of 1.

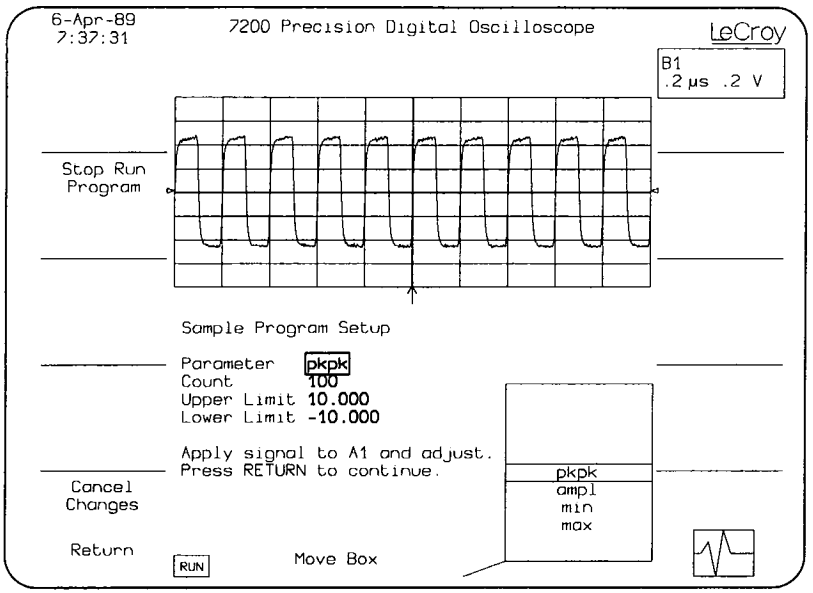


Figure 7.1 Sample Program Setup

In this case, there are 9 lines, so the upper half of the screen will contain a grid for displaying trace 1. This allows the user to see the data while the menu is active.

While a menu is active, front panel controls are available for adjusting acquisition parameters. This fact is used in this menu to allow the user to configure the input top provide a good view of the wave form.

Figure 7.1 shows the 7200A's screen during this menu.

85 The title of the menu, "Sample Program Setup" is indicated by this display statement.

86 This line provides a blank line between the title and the rest of the menu.

87 The var statement is used to allow selection of a value for the variable "parm" from the list "parameters", defined in line 8. As in other setup screens in the 7200A, values are selected by rotating the continuous knob below the center of the screen. The detented knob is used to move the cursor box from one choice to another, as defined by the *var* statements in the menu. The string after the word *display* is used to lable the choice.

88 This statement is used to set the variable "count" to be an integer between 1 and 5000. This is the number of waveforms which will be sampled by the main loop, lines 13 to 20.

89-90 These var statements set the variables "high" and "low" to real values in the range -10 to +10, with a precision of 0.001, or 1 mV.

91-93 The remaining lines are a message to the user. Note that this message must be kept short so the number of lines in the menu is less than 14. This is necessary to allow display of a trace in the upper half of the screen.

94 This line indicates the end of the menu "set_up".

96-99 This is the beginning of the procedure "acquire".

100-102 The *status clear* statement is used to clear the indication that processing is complete for trace 1. This is done before acquiring another waveform to make sure that the wait statement in line 108 waits for the next acquisition, and not a previous one.

104-105 The "ARM" command indicates a single acquisition. When acquisition is complete, processing as defined in line 35 is started. When the processing is complete, the status "PROCESSING DONE 1" is set.

107-108

This statement waits for any enabled status condition to be set. In this case, only one status is enabled: "PROCESSING DONE 1". If the condition is not set within 30 seconds, the *wait* statement finishes and the next statement is executed.

Note that there is another variation, *wait for all status*, which waits for all enabled status conditions to be set. For example. if the application required processing for traces 1 and 2, we would enable both conditions "PROCESSING DONE 1" and "PROCESSING DONE 2".

110-117

These lines begin with a test to see if the *wait* statement timed out. This is done using the *cond* function, which determines if a particular condition caused the *wait* statement to finish, as described on p. 7.53.

If the "TIMEOUT" condition is set, the cond function returns true. This causes the statements prior to the *endif* in line 117 to be executed.

Line 113 prints a message near the top of the display to indicate that a trigger was not received within 30 seconds.

Line 116 uses a "PROG_MODE" command to stop the program. This command, described in section 5, is normally used by a remote computer to control the operation of an ICL program in the 7200A. However, it can also be used by an ICL program to halt itself, since setting the mode to "off" has that effect.

118 This line indicates the end of the procedure "acquire".

120-124 This is the beginning of the procedure "update".

125-128 This statement uses the query function to determine the value of a waveform parameter. The argument is a string composed by joining the string 'TI:PAVA?' with the name of the parameter. This query, "PARAMETER_VALUE", is described in section 5. Almost any query described in section 5 or 6 of this manual may be given. See p. 7-80 for more details on the query function.

The resulting string is stored in the variable "value". For the parameter "pkpk", this string might be 'PKPK,1.5 V,OK'. the structure of this string is used by the search and token functions in the next few lines of the program.

130 This if statement looks for the string 'OK' in the query result. If present, lines 131 to 143 are executed. If not, lines 145 and 146 are executed instead. This field in the parameter result is called the "state". It is used to indicate the quality of the result. The value 'OK' indicates that the parameter was computed without any problems.

133-134 The token function is used to extract the number from the string returned by query in line 128. The second argument to token is the number of the token to be extracted. Tokens include words, numbers, and other single characters. In the example above, "PKPK" is the first token, the comma is the second, "1.5" is the third, and so on.

The number extracted by the token function is stored back into the same variable, "value". Note that "value" was a string when assigned in line 128, but now it is a number. Variables in ICL are not of any fixed type, such as integer or string. Instead, they take on the type of the quantity which is stored in them. When necessary, they are converted from one type to another, as described in the section "Automatic type conversion" on p. 7-24.

136-143	These statements use the *if, elseif, else* and *endif* to determine the counter to be incremented depending on the value of the waveform parameter. If the relation in the *if* statement is true, the statements prior to the *elseif* are executed. If not the relation following *elseif* is tested, If it is true, the statements prior to the *else* are executed. If that relation is false, the statements between the *else* and the *endif* are executed.
144	This else statement ends the statements which are executed if the string 'OK' was found in the query result ion line 130.
145-146	These lines add 1 to the variable "invalid" to indicate that the waveform parameter was not valid.
147	This else statement ends the statements which are executed if the waveform parameter was invalid.
148	This line indicates the end of the procedure "update",.
150-154	This is the beginning of the menu "report". This menu is used only to present information to the user. It does not contain any var statements for adjusting the values of variables.
155	This line gives a title for the report.
156	This display statement leaves a blank line after the title.
157-160	These statements use the format function to produce a tabularized list of the results. The first argument is a string which controls the location and format of the values printed. The remaining arguments supply the values to be printed. The format function is described in detail on p. 7-69.
163-165	The remaining lines are a message to the user. Note that this message must be kept short so the number of lines in the menu is less than 14. This is necessary to allow display of a trace in the upper half of the screen.
166	This line indicates the end of the menu "report".

Elements of a Program

A program consists of many components such as constants, variables, expressions, commands, and statements. If you already have experience with a programming language, most of the components will be familiar.

Numeric Constants

ICL supports two different types of numbers: integers and real numbers. Integers are strings of the digits 0 to 9, possibly preceded by a minus sign. The allowed range of an integer is -2147483648 to 2147483647.

Integers may be expressed in different number bases by preceding them with a pound sign (#) followed by "H" for hexadecimal, "Q" for octal, or "B" for binary. The prefix "Ox" is also accepted to denote hexadecimal, as in the C programming language. For example, #HFF, #Q377, and #B11111111, and)xFF are all 255.

Real numbers are used to represent very large numbers, or numbers with fractional parts. Some examples are 1.23, -200.01, 2E-5 and 6.02e23. The last two examples use scientific notation. The number following the "E" or "e" os the power of ten, and may be negative. Real numbers have approximately seven significant digits, with an allowed exponent range of -38 to 38.

Strings

A string is a sequence of characters enclosed in single quotes. A quote may be included in a string by placing two quotes next to each other. Double quotes are not used by ICL defining strings. Some examples of strings are "hello" and "You don't say!"

Identifiers

Identifiers are used for naming variables and procedures (subroutines). An identifier is any sequence of letters, digits, and underscores, provided that the first character is not a digit.

ICL makes no distinction between upper case and lower case letters in identifiers. Therefore, the following identifiers are equivalent: Smallest_Value, SMALLEST_VALUE, and smallest_value.

Note that upper vs. lower case is used to distinguish statements from commands, discussed below. Sometimes you will be required to begin an identifier with a lower case letter (as in an assignment statement), and sometimes you will be required to begin it with an upper case letter (as in command).

ICL does not limit the number of characters in an identifier. However, it is a good idea to keep the identifier shorter than a line on your editor's screen, since most statements are required to fit on a single line.

Variables

Variables are used to hold values. They can hold integer or real numeric values, or strings. They cam also take on values defined by the *list* statement, described on p. 7-44.

Variables are not declared in ICL. Instead, they are recognized by their use. Any statement which can change the value of a variable, such as an assignment or an input statement, defines a variable.

Variables are global in scope. That is, if the same name is used for a variable in more than one part of the program, each use refers to the same variable. This includes uses of variables in different procedures.

The type of a variable is not fixed, but may change whenever its value changes. Thus, it is possible for a single variable to have the values -1, 3.5, and "error" at different times during execution of a program.

Before a value is assigned to a variable, it is "undefined". Attempting to use the value of an undefined variable will result in an error message and program termination.

Arrays

Arrays are used to hold collections of values. Like variables, they can hold integer or real numeric values, strings, or values defined by list statements.

Arrays are declared using the array statement, which defines the type of values which can be stored as well as the number and range of subscripts.

Each array may be declared only once in a program. The array statement must appear before any uses of array. The scope of an array is from its declaring array statement to the end of the program.

When an array is declared it is filled with values depending on its type. Integer and real arrays are filled with zeros. List arrays are filled with the first element of the list. String arrays are undefined. An attempt to use a string array element before storing a value in it will result in an error message and program termination.

Elements of an array can be used in most places where variables can be used. An array element is indicated by the name of the array, followed by a series of subscript expressions enclosed in square brackets and separated by commas, as in data [i,j+5]. If the number of subscripts does not agree with the declaration for the array, an error message is given and compilation is terminated. A subscript out of range results in a run-time error and execution is terminated.

Expressions

Expressions allow you to combine values (in constants, variables, and array elements) to form new values. Expressions in ICL are very similar to those found in other programming languages.

Numeric Operators

The arithmetic operators addition, subtraction, multiplication, and division are represented by the usual symbols: +,-,*,and ?. Division by zero is not permitted. Note that division always produces a real result, and never truncates to the next lower integer. The trunc function may be used for that purpose if necessary. See p. 7-91 for more information.

The power operator, ^, raises one number to the power of another number. 5 ^ 3 is the same as 5 * 5 * 5, or 125. If the first operand is zero, the second must be greater than zero. If the first operand is negative, the second must be an integer.

The mod operator returns the remainder which is left after dividing one number by another. 10 mod 3 is 1, since 10/3 is 3 with a remainder of 1.

Several operators are available for logical combination of values. These operators act bit-wise on integer values. That is, each binary digit in the two values is combined independently to produce a binary digit in the resulting value.

The *and* operator produces a binary digit 1 wherever both values have binary digits 1 in the same position. It produces a binary digit 0 in all other cases, as illustrated in the following examples:

```
0 and 0 = 0
0 and 1 = 0
1 and 0 = 0
1 and 1 = 1
#B0011 and #B0101 = #B0001
```

The *or* operator produces a binary digit 0 wherever both values have binary digits 0 in the same position. It produces a binary digit 1 in all other cases, as illustrated in the following examples:

```
0 or 0 = 0
0 or 1 = 1
1 or 0 = 1
1 or 1 = 1
#B0011 or #B0101 = #B0111
```

The *xor* operator (exclusive or) produces a binary digit 1 wherever the values have different binary digits in the same position. It produces a binary digit 0 if the values have the same binary digit, as illustrated in the examples below. Note that a value may be ones complemented using an exclusive or with #HFFFFFFFF.

```
0 xor 0 = 0
0 xor 1 = 1
1 xor 0 = 1
```

```
1 xor 1 = 0
#B0011 xor #B0101= #B0110
```

String Concatenation Operators

Two strings may be concatenated (joined) using the "|" or "||" operators. The result of 'Hello'|'world' is 'Helloworld'.

You might want to join two strings together with a space between them. In the preceding example, 'Hello world' could be produced by first concatenating 'Hello' and ' ', and then concatenating the result with 'world', as in 'Hello'|' '|'world' .

The "||" operator makes this easier. It joins the two strings together with a single space between them. The result of 'Hello' || 'world' is 'Hello world'.

Relational Operators

Relational operators compare values in order to make decisions in a program. These operators compare two values and produce a result which is either TRUE (represented by 1) or FALSE (represented by 0).

The standard relational operators are the same as in the C language:

a != b	TRUE if a is not equal to b
a < b	TRUE if a is less than b
a <= b	TRUE if a is less than or equal to b
a = b	TRUE if a is equal to b
a > b	TRUE if a is greater than b
a >= b	TRUE if a is greater than or equal to b

These operators compare two values of the same type: integer, real, or string. For integer and real values, the meanings are familiar. For strings, normal dictionary ordering is used, except that all upper case letters are considered to be "less than" all lower case letters.

In cases where the two values are different, they are first converted to the same type according to the following rules:

- If one is numeric and the other is string, the string is checked to see if it contains a number. In this case, the string is converted to a number.
- If one is numeric and the other is string, but the string does not contain a number, the numeric value is converted to a string as it would be for printing.
- If one is integer and the other is real, the integer value is converted to real.

If a comparison must be done numerically, adding 0 to both arguments will ensure that they are numbers. Similarly, if a string comparison is required, concatenate an empty string. For example:

a+0 <= b+0	forced numeric comparison
a|'' = b|''	forced string comparison

In addition to the standard relational operators, ICL has a set of approximate relational operators which are useful for determining if two real numbers are almost the same, or if two strings are the same except for character case and spacing:

a ~! b	TRUE if a is approximately not equal to b
a ~< b	TRUE if a is approximately less than b
a ~<= b	TRUE if a is approximately less than or equal to b
a ~= b	TRUE if a is approximately equal to b
a ~> b	TRUE if a is approximately greater than b
a ~>= b	TRUE if a is approximately greater than or equal to b

Like the standard relational operators, the approximate relational operators compare two values of the same type: integer, real, or string. The same rules are used to convert the types of the values if necessary.

The meaning of "approximate" depends on the type of the values. For integers, there is no difference between the approximate and standard relational operators. For reals, two values, a and b, are considered approximately equal if the absolute value of (a-b)/(a+b) is less than or equal to 0.000001.

Strings are compared ignoring differences in character case and number of spaces by first converting both strings to a standard form, and then performing the corresponding standard comparison. The conversion changes lower case letters to upper case, removes leading and trailing whitespace, and changes each sequence of multiple whitespace characters to a single space. This conversion may also be performed manually with the built-in function *upper*, described on p. 7-93.

Order of Operations

When an expression involves more than one operator, you can control the order in which the operators are applied using operator precedence. The operators are listed from highest precedence to lowest precedence in the table which follows.

highest	Power operator	^
	Multiplying operators	* / mod and
	Adding operators	+ - or xor
	String operators	| ||
lowest	Relational operators	! = < < = = > > = ~ ! = ~ < ~ < = ~ = ~ > ~ > =

Expressions are evaluated by applying operators in order from highest to lowest precedence. For example, in the expression 5 + 2 * 3, the multiply is done before the add, resulting in the value 11. Parentheses may be used to control the order of evaluation. For example, the expression (5+2) * 3 indicates that the addition is to be performed first. In this case, the result is 21.

When several operators of equal precedence appear in an expression, they are applied left to right. For example, 5 + 3 - 2 - 1 is equivalent by adding 5 + 3, then subtracting 2, and finally subtracting 1, with a result of 5.

Built-In Functions

ICL provides built-in functions to compute many different values. Real to integer conversions, trigonometric functions, and string search and extraction functions are among those available. See p. 7-55.

Automatic Type Conversion

Automatic type conversion eliminates the need to change values explicity from one type to another. This concept, introduced above in the discussion of relational operators, is used extensively in ICL.

Inputs to functions and operators, values to be stored in array elements, and array subscripts must be of a particular type. If the value actually supplied is not already of the correct type, it is converted according to the following rules:

- If a string is supplied and an integer or real is required, the string is examined to see if it contains a number. If the string contains a number, it is converted to a number. If it can't be converted to a number, an error is reported and the program is terminated.
- If an integer is supplied and a string is required, the number is converted to a string just as it would be for printing.
- If an integer is supplied and a real is required, the integer is converted to the corresponding real value.
- If a real is supplied and an integer is required, the real value is truncated to the nearest integer which is closer to zero. (see the discussion of the *trunc* function, p. 7-91 for more details.)

Commands and Statements

An ICL program consists of lines of text. With few exceptions, each line contains one command or statement. (The exceptions are blank lines, which are ignored, and *list* statements, which may span several lines.) Keep each line short enough to fit onto one line of your editor's screen.

A command instructs the 7200A to set operating conditions or initiate actions. The commands are described in sections 5 and 6 of this manual. In addition to appearing in an ICL program, a command may be sent through the GPIB or RS-232 interface.

A statement controls the sequence of commands executed by an ICL program. Statements are described in the following section. They cannot be given to the 7200A through the GPIB or RS-232 interface.

Statements are distinguished from commands by the first non-blank character of the line in which they appear. Statements always start with a lower case letter. Commands start with an upper case letter or an asterisk(*).

Command Preprocessing

The 7200A processes commands as though they were received via GPIB or RS-232. Before the normal processing begins, however some additional preprocessing takes place when a command originates in an ICL program.

The 7200A scans each command, looking fro opportunities to make substitutions. If a variable or array element is found in the command, its current value is substituted. If an expression is found, it is evaluated, and the result is substituted.

Numeric constants by themselves are not substituted. For example, 1.00000 is never changed to 1, even though it is shorter or might be considered more correct.

Because of command preprocessing, you will want to choose your variable names carefully. For example, you may want to generate the command.

```
GRID SINGLE
```

However, if you have a variable called SINGLE, the currect contents of the variable will be substituted. To prevent this, simply place the word SINGLE in quotes:

```
GRID 'SINGLE'
```

Since double quotes are not used by ICL in defining strings, they may be used in commands without affecting substitutions. For example:

```
equation = 'A1 +A2'
T1:DEF EQN,"equation"
```

This sequence assigns the strings 'A1 + A2' top the variable "equation". Since double quotes are used in the command defining the equation for trace 1, substitution occurs to produce the command

```
T1:DEF EQN,"A1 + A2"
```

Procedures and Menus

A procedure performs actions. A procedure begins with a *procedure* statement and ends with an *end* statement. Within a procedure, all commands and statements except the *display* and *var* statements may appear.

A menu describes a setup screen which is used wby the operator to change values of variables and initiate actions. A menu begins with a *menu* statement and ends with an *end* statement. Within a menu, only the *call, display*, and *var* statements are allowed.

The main program appears at the beginning of the source file, consists of a series of commands and statements, and ends with and *end* statement. Procedures and menus appear after the main program in any convenient order.

Procedures and menus can be called from the main program, or other procedures or menus.The one exception is that a procedure should not call itself, directly or indirectly. this technique, called recursion, is not generally of value in a language in which all variables are global.

Comments

Comments begin with a semicolon (;) and continue to the end of the line. A comment may appear on any line of the program, and may either occupy an entire line or part of a line to the right of a command or statement. Comments have no affect on the program, and if particular do not make the program execute more slowly.

Blank lines and initial whitespace (indention) may be inserted as desired to make the program easier to understand.

Statements

The statements provided by ICL are listed by category in the following table:

category	statement	purpose
expressions	array	define an array
	list	define an enumerated list
	assignment	assign a value to a variable
flow of control	call	call a procedure or menu
	for...endloop	iteration over a set of values
	while...endloop	iteration while a condition is true
	break	early termination of *for* or *while* loop
	if...elseif...else...endif	conditional execution
	procedure	define a procedure
	return	return from procedure
	end	end of program, menu, or procedure
input/output	input	input text from host computer
	print	print to display/host/hardcopy
events	status	GPIB status enable/disable/clear
	wait	wait for status change or time interval
interaction	menu	define a menu for user interaction
	call	attach a procedure or menu to a key
	display	add a line of text to a menu
	var	add a variable to a menu for modification
comment	; to end of line	documentation

The following sections describe each statement in detail. Statements are listed alphabetically. With the exception of the *list* statement, each statement must fit on a single line.

array

Purpose: Declare an array.

Format: *array* type array_name *[* expr*:* expr, expr*:* expr...*]*

Type is *integer*, *real*, *string*, or the name of a list previously defined in the program . Each element of the array is of the indicated type.

The range of each subscript is given by two expressions separated by a colon. Subscripts are integer values. If real values are used, they are truncated toward zero.

The number of subscripts is defined by the number of pairs of expressions. The correct number of subscripts must be used when an element of the array is referenced.

Examples:

```
array integer data [0:5, -10:10]
```

The array data is defined to be of type integer, having two subscripts. The first subscript can take one of 6 values: 0 through 5. The second can take on of the 11 values: -10 through 10. Therefore there are 66 elements in the array.

```
list colors is (red orange yellow green blue violet)
array colors select [0:3]
select [3] = next (select [0])
```

The array select has four elements, each from the list of colors. All four elements are set by the array statement to the value red, the first entry in the list. The assignment then sets the last element to orange, using the *next* function and the current value of the first element.

```
arrray real hue [ord (colors, red) : ord(colors, violet)]
for h in colors
      hue [ord(colors, h)] = 2.43
endloop
```

This example uses the *ord* function to define the range of subscripts for an array. In this case, the array ccontains one element for each entry in the list colors. The loop uses the *ord* function to set each element of the hue array to 2.43.

array (continued)

Notes:

When an array is first declared, it is filled with values depending on its type. Integer and real arrays are filled with zeros. List arrays are filled with the first element of the list. String arrays are undefined. An attempt to use a string array element before storing a value in it will result in an error message and program termination.

Only one array statement may appear for each array in the program. However, an array statement may be executed more than once. This would happen if it is in a loop or in a procedure which is called more than once.

If an array statement is re-executed with identical subscript expressions, no change is made to the array. If the subscript expressions are different , the array reinitialized. In this case, all previous values are lost.

Array elements can be changed only by assignment statements. This means that an array element cannot be used in an *input* statement, or as the loop variable in a *for* statement.

An array element may be used to compute any expression wherever a variable may be used. Array elements may also appear in commands, and are replaced by their current value before passing the command to the 7200A for further processing.

See Also:

list statement, built-in functions *next* and *ord*

assignment

Purpose: Assign a value to a variable. The value may be any valid expression as described in the section "Elements of a program".

In addition to assigning a value to the variable, this statement also sets the type of the variable to that of the expression.

Format: variable = expression

Examples:

```
i = 3
next_value = current_value + 1
y = 5.0 * sin(x * pi / 180)
error_description = 'invalid value'
```

Notes: This statement also serves to define the variable.

The variable must begin with a lower case letter in this case to distinguish the assignment statement from a GPIB command.

break

Purpose: Terminate execution of a loop immediately.

Format: *break*

Examples:

```
for i = 1 to 100
  call measure_something
  if error_was_detected
    break
  endif
endloop

while i < 100
  call measure_something
  if error_was_detected
    break
  endif
endloop
```

Note: After the *break*, the next statement which is executed is the first statement after the endloop of the innermost loop which contains the *break*.

If a *break* statement is used to terminate a *for* loop, the iteration variable is left at the value it had when the *break* was executed.

See Also: *for, while*

call

Purpose:

Execute a menu or procedure.

Install a menu or procedure in a menu to allow the operator to execute the menu or procedure by pressing a key.

Format:

call name
call name *key* k *display* expression

The first form of the *call* statement is used in the main program or a procedure. The procedure or menu is executed immediately.

The second form of the *call* statement is used in a menu. The procedure or menu is executed when the operator presses the indicated key.

Name is the name of any menu or procedure.

Keys 0 to 9 are on the left side of the display, with 0 at the top. Keys 10 and 11 are on the right side of the display, with 11 at the bottom.

The expression following the word display is evaluated as a string to determine the label to write next to the key. It should be no more than 13 characters in length.

Examples:

```
call measure_something

call procedure_to_reinitialize key 1 display 'Initialize'
call menu_to_set_size key 2 display 'Set Size'
```

Note:

When a procedure or menu is called from the main program or another procedure, the next statement after the *call* statement is executed upon return.

When a procedure or menu is called from a menu, the menu is re-executed upon return.

The "*display* expression" clause is optional. If it is not present, the name of the menu or procedure is used instead.

See Also:

menu, procedure, return, var, call

comment

Purpose: Provide documentation for a program. A comment begins with a semicolon and continues to the end of the line.

Format: ; any text

Examples:

```
; This comment is an entire line.

x = 50 * I ; Set x to the expected value.
```

Note: A semicolon enclosed in quotes does not start a comment.

display

Purpose: Add a line of text to a menu for documentation purposes.

Format: *display* string_expression

Examples: **display 'Test Set Up'**

Note: If the string_expression is not given, a blank line is displayed.

The number of lines in a menu determines the format of the screen. Each *display* statement adds one line to the display. Each *var* statement without the *key* option also adds one line to the display.

If there are no lines defined in the menu, only the key labels are replaced, and the full screen is available for the display of traces. If there are 1 to 13 lines in the menu, the menu occupies the lower half of the screen, and the upper half of the screen is available for traces. If there are 14 or more lines defined in the menu, the menu occupies the entire screen.

See Also: *menu, var*

else

Purpose: Indicate the statements to be executed if the expression of an *if* statement is false.

Format: *else*

Examples:

```
If expression
  ; statements executed if the expression is true
else
  ; statements executed if the expression is false
endif

If expression1
  ; statements executed if expression1 is true
elseif expression2
  ; statements executed if expression2 is true and expression1 is
false
else
  ; statements executed if both expressions are false
endif
```

See Also: *if, elseif, endif*

elseif

Purpose: Indicate a new test and statements to be executed if the expression of an *if* statement is false.

Format: *elseif*

Examples:

```
if expression1
  ; statements executed if expression1 is true
elseif expression2
  ; statements executed if expression2 is true and expression1 is
false
else
  ; statements executed if both expressions are false
endif
```

This example is the same as

```
if expression1
  ; statements executed if expression1 is true
else
  if expression2
    ; statements executed if expression2 is true and expression1 is
false    else
    ; statements executed if both expressions are false
  endif
endif
```

See Also: *if, else, endif*

end

Purpose: Indicate the end of a program, procedure, or menu.

Format: *end*

Examples:

```
procedure measure_something
  ; Perform a measurement and store
  ; the result in a variable.
end
```

Note: The main program always appears before any procedures or menus, so an *end* statement must appear before the first procedure or menu.

See Also: *menu, procedure*

endif

Purpose: Indicate the end of an *if* statement.

Format: *endif*

Examples:

```
If expression
  ; statements executed if the expression is true
else
  ; statements executed if the expression is false
endif

If expression1
  ; statements executed if expression1 is true
elseif expression2
  ; statements executed if expression2 is true and expression1 is false
else
  ; statements executed if both expressions are false
endif
```

See Also: *if, else, elseif*

endloop

Purpose: Indicate the end of a *for* or *while* loop.

Format: *endloop*

Examples:

```
for i = 1 to 100
  ; statements
endloop
```

See Also: *for, while*

for

Purpose:

Repeatedly execute a series of statements. The *for* loop uses a control variable which takes on different values on each iteration of the loop.

Format:

for variable = expression1 *to* expression2
for variable = expression1 *to* expression2 *step* expression3
for variable *in* list_name

In the first two forms of the *for* statement, the expressions should be numeric. In the first iteration, the control variable is set to expression1. On successive iterations, the *step* value is added to the control variable. The loop terminates when the control variable is greater than expression2. The *step* value is 1 in the first form of the *for* statement, and expression3 in the second form.

In the third form of the *for* statement, the control variable takes on the values of the elements of the indicated list. In the first iteration, it is set to the first element of the list. In the second iteration, it is set to the second element of the list, and so on.

Examples:

```
for i = 1 to 100
 ; statements
endloop
```

```
for i = 1 to 100 step 10
 ; statements
endloop
```

```
list traces is (T1 T2 T3 T4 T5 T6 T7 T8)
for trace in traces
 Trace: TRA OFF
endloop
```

Note:

This statement also serves to define the control variable.

In addition to assigning values to the control variable, this statement also sets the type of the control variable.

for (continued)

When real expressions are used, care should be taken to ensure that rounding error does not cause unexpected results. For example, consider the following statement:

```
for i = 1 to 2 step 0.001
```

The variable i is expected to take on the values 1.000, 1.001, 1.002, ..., 1.999, 2.000. However, rounding error involved in the real additions might make the 1001st value be 2.000001. In this case, since the control variable is greater than the final specified value, the loop would not execute the last time. A better way to write this loop would be:

```
for i = 1 to 2.0005 step 0.001
```

See Also: *break, endloop, list, while*

if

Purpose: Select statements for execution.

Format: *if* expression

Examples:

```
if expression
  ; statements executed if the expression is true
endif

if expression
  ; statements executed if the expression is true
else
  ; statements executed if the expression is false
endif

if expression1
  ; statements executed if expression1 is true
elseif expression2
  ; statements executed if expression2 is true and expression1 is
false
else
  ; statements executed if both expressions are false
endif
```

Note: The expression in an *if* statement may be the result of a relational operator (described in the section "Elements of a Program") or may be any numeric expression. For numeric expressions, "true" means non-zero, and "false" means zero.

See Also: *else, elseif, endif*

input

Purpose: Read from an external device. Data is read from the specified device and transferred to the indicated variable.

Format: *input* variable *from* device

Device is either *host* or *hardcopy*. *Host* indicates a computer connected via GPIB or RS-232, as determined in the Communication Setup screen. *Hardcopy* indicates the device selected in the Hardcopy Setup screen. Note that input from the centronics parallel port is not supported.

Examples: **Input val from host**

Note: This statement also serves to define the variable.

The data is initially treated as a string. However, if the contents permit, it may be used as a numeric value, since the automatic type conversion process will be applied.

See Also: *print*

list

Purpose: Define a list. A list is similar to an enum type in C or a scalar type in Pascal. It consists of a series of items delimited by whitespace. The elements of the list may be identifiers, numeric constants, or strings.

Format: *list* list_name *is (* element1 element2 element3 ... elementN *)*

Examples:

```
list colors is (red orange yellow green blue violet)

list allowed_values is (1 2 5 10 20 50 100 200 500 undefined)

list error_messages is (
   no_error
   'value too large'
   'value too small'
   'value is not a number'
)
```

Note: An identifier used in a list may not be the name of a variable, array, procedure, or menu defined by the program, and may not be the same as any keyword or built-in function used by the language.

The same entry may appear in more than one list, but may not appear more than once in the same list.

The *list* statement may span several lines. However, the name of the list, the keyword *is*, and the left parenthesis must be on the first line.

Elements in a list need not all be of the same type. Some may be numeric, others may be identifiers, and still others may be strings. The *type* built-in function may be used to determine what type of element has been assigned to a variable before it is used.

See Also: *for*, *var*, built-in functions *first, last, next, ord, prev, type*

menu

Purpose: Begin definition of a menu.

Format: *menu* name

Examples:

```
; This is the main program.
 list test_list is (random linear quadratic)
 call set_up_test
end

menu set_up_test
 display 'Test Set Up'
 display
 var t list test_list                  display 'Test to perform    '
 var n integer 1 to 100                display 'Number of times    '
 var high real 0 to 10.0 step 0.01     display 'high               '
 var low real 0 to 10.0 step 0.01      display 'low                '
end
```

Note: Only *menu* statements (*call*, *display*, and *var*) can appear in a menu. A menu is terminated with an *end* statement.

The *menu* returns when the "Return" or "Cancel Changes" key is pressed.

While a menu is being displayed, all front-panel controls are available for use. This allows adjustment of acquisition and display parameters. However, entering a setup screen (e.g. by pressing "DISPLAY" on the 7242) will terminate the menu and deactivate the front panel.

See Also: *call, procedure, return, display, var*

print

Purpose: Print a string to an external device or the display.

Format:
print device expression
println device expression

The first form writes the expression without an end of line character. The second form writes an end of line character after the expression.

Device is either *host*, *hardcopy*, or *display*. *Host* indicates a computer connected via GPIB or RS-232, as determined in the Communication Setup screen. *Hardcopy* indicates the device selected in the Hardcopy Setup screen. *Display* indicates a one-line message area at the top of the CRT.

Examples:

```
x = 10
print display 'The value is' || x     prints The value is 10
println hardcopy 'YES'                prints YES
```

See Also: *input*

procedure

Purpose: Begin definition of a procedure.

Format: *procedure* name

Examples:

```
procedure waste_time
 for i = 1 to 1000
 endloop
end
```

Note: A procedure should not call itself, either directly or indirectly.

Values may be passed to and from procedures by using variables.

See Also: *menu, call, return*

return

Purpose: Return from a procedure.

Format: *return*

Examples:

```
procedure measure_something
  ; statements
  if error_occurred
    return
  endif
  ; statements
end
```

Note: A *return* statement may only appear in a procedure, and is not necessary at the end of a procedure.

See Also: *call, procedure*

status

Purpose: Enable, disable, or clear status conditions.

Status conditions indicate the state of the 7200A, and are a subset of the GPIB status bytes defined in section 4 of this manual.

Conditions supported by the *status* statement are:

AUTO SETUP DONE	Auto Setup has completed.
CAL DONE plugin	Calibration has completed for the plugin indicated. Plugin is one of A or B.
HARDCOPY DONE	Hardcopy has completed.
MAX SWEEPS trace	The maximum number of sweeps set for a history function has been accumulated on the indicated trace. Trace is one of 1, 2, 3, 4, 5, 6, 7, or 8.
PROCESSING DONE trace	Processing has completed for one display update of the indicated trace. Trace is one of 1, 2, 3, 4, 5, 6, 7, or 8.
RECALL DONE	Recall from floppy disk has completed.
RECORD DONE	Store to disk during Record Traces has completed.
REPLAY DONE	Recall from disk during Replay Traces has completed.
SELF-TEST DONE	Self testing has completed.
STORE DONE	Store to floppy disk has completed.
TRIGGER plugin	Data has been acquired from the indicated plugin. Plugin is one of A or B.

status (continued)

Format:

status enable condition
status disable
status disable condition
status clear
status clear condition

Condition is a string expression which indicates one of the conditions described above.

Examples:

The following is an example of the recommended order in which *status* statements should be used to trigger an acquisition and wait for processing done on all traces.

```
list processing_done is (                ; define the list of
   'PROCESSING DONE 1'                   ; processing done status
   'PROCESSING DONE 2'                   ; required for the problem
   'PROCESSING DONE 3'
   'PROCESSING DONE 4'
   'PROCESSING DONE 5'
   'PROCESSING DONE 6'
   'PROCESSING DONE 7'
)
STOP                                     : stop pending acquisitions
                                         ; and go to single trigger mode
status disable                           ; disable all status
status clear                             ; clear all status
for c in processing_done                 ; enable all processing
 status enable c                         ; done status
endloop
ARM                                      ; arm all input channels
wait for all status or 30 seconds        ; wait for all enabled status
                                         ; or a timeout of 30 seconds
```

See Also:

wait, built-in function *cond*, Section 4: Status Messages

var

Purpose: Install a variable in a menu to allow the operator to set the value during execution of the menu.

Format:

var name variable_info *display* expression
var name variable_info *key* k *display* expression
var name variable_info *key* k1 *and* k2 *display* expression

The first form of the *var* statement places the variable in the main body of the menu. This allows the operator to change the value using the cursor knobs below the display.

The other forms of the *var* statement place the variable next to a softkey. Keys 0 to 9 are on the left side of the display, with 0 at the top. Keys 10 and 11 are on the right side of the display, with 11 at the bottom.

variable_info is one of:

string
list list_name
integer expression1 *to* expression2
real expression1 *to* expression2 *step* expression3

The variable_info is used to describe the type of variable and its possible values. The *string* form makes the variable a string. When the cursor box is moved onto the variable's value, the "Edit Text" softkey appears in the upper left corner of the display. Pressing this key allows you to edit the string in the same manner as for other strings in the 7200A.

The *list* variable_info form means the variable is to be a value from the named list.

The *integer* variable_info form makes the variable an integer. The range of values is indicated by expression1 (the lower limit) and expression2 (the upper limit).

The *real* variable_info form makes the variable a real number. The range of values is indicated by expression1 (the lower limit) and expression2 (the upper limit). Expression3 indicates the resolution of the value. For example, if expression3 were 0.01, the value would be displayed with two digits to the right of the decimal point.

var (continued)

Only list and integer variables may be attached to keys. If only one key is assigned to the variable, the variable is increased each time the key is pressed. When the maximum value is reached, the next key press set the variable to the minimum value.

If two keys are assigned to a variable, the first key increases the value, and the second key decreases the value. In this case, the first key should be even numbered, and the second key should be one greater than the first key.

The expression following the word *display* is evaluated as a string to determine the label to write in the menu or next to the key. Key labels should be no more than 13 characters in length.

Examples:

```
var t list test_list                    display 'Test to perform    '
var n integer 1 to 100                  display 'Number of times    '
var high real 0 to 10.0 step 0.01       display 'high               '
var number integer 1 to 10 key 0 and 1
```

Note:

When the menu is entered, variables which are attached to the menu are examined. Any undefined variables, or variables which currently have values which can't be converted to the type indicated in the *var* statement are assigned a value. For lists, this is the first element in the list. For integers and reals, it is the lower limit. For strings, it is an empty string.

The "display expression" clause is always optional. If it is not present, the name of the variable is used instead.

The number of lines in a menu determines the format of the screen. Each *display* statement adds one line to the display. Each *var* statement without the *key* option also adds one line to the display.

If there are no lines defined in the menu, only the key labels are replaced, and the full screen is available for the display of traces. If there are 1 to 13 lines in the menu, the menu occupies the lower half of the screen, and the upper half of the screen is available for traces. If there are 14 or more lines defined in the menu, the menu occupies the entire screen.

See Also:

menu, display, call

wait

Purpose: Wait for conditions and/or a predetermined time.

Format:
wait for any status
wait for any status or expression *seconds*
wait for all status
wait for all status or expression *seconds*
wait for expression *seconds*

Examples:
wait for any status
wait for any status or 30 seconds
wait for all status
wait for all status or 1.5 seconds
wait for 0.2 seconds

Note: The expression must be a numeric quantity. If it is less than or equal to zero, a small delay will occur.

The delay indicated by the expression is approximate.

See Also: *status*, built-in function *cond*

while

Purpose: Repeatedly execute a series of statements. The expression is evaluated before each iteration. If it is true, the statements preceding the matching *endloop* are executed. If it is false, execution continues with the statement after the matching *endloop*.

Format: *while* expression

Examples:

```
while I < 100
  I = I + J
endloop
```

Note: The expression in a *while* statement may be the result of a relational operator (described in the section "Elements of a program") or may be any numeric expression. For numeric expressions, "true" means non-zero, and "false" means zero.

A *break* statement may be used to terminate a *while* loop.

See Also: *endloop, break*

Built-in Functions

The functions provided by ICL are listed by category in the following table:

category	function	purpose
math	abs	absolute value of a numeric quantity
	exp	value of e raised to the power of the argument
	log	natural logarithm of a numeric quantity
	log10	base 10 logarithm of a numeric quantity
	sign	sign of a numeric quantity
	sqrt	square root of a numeric quantity
trig	acos	inverse cosine of a numeric quantity
	asin	inverse sine of a numeric quantity
	atan	inverse tangent of a numeric quantity
	atan2	inverse tangent of a ratio
	cos	cosine of an angle expressed in radians
	sin	sine of an angle expressed in radians
	tan	tangent of an angle expressed in radians
conversion	ceil	next higher integer
	chr	convert a numeric value to an ASCII character
	floor	next lower integer
	round	round a real value to the nearest integer
	trunc	truncate a real value to an integer
string	format	format a string
	left	substring from left of a string
	mid	substring from middle of a string
	right	substring from right of a string
	search	search a string for a substring
	strlen	length of a string
	token	extract a token from a string
	upper	convert a string to upper case
list	first	first element in a list
	last	last element in a list
	next	next element in a list
	ord	ordinal number of element in a list
	prev	previous element in a list

Built-in Functions (continued)

category	function	purpose
general	cond	tests a status condition
	query	query system information
	type	information about an expression or variable

The following sections describe each function in detail. Functions are listed alphabetically

abs

Purpose: Compute the absolute value of a numeric quantity.

Format: *abs(x)*

Argument: any numeric quantity

Return value: The absolute value of the argument is returned.

Return type: The type of the return value is the same as the type of the argument. If the argument is an integer, the result is an integer. If the argument is a real, the result is a real.

Examples:

abs(x) returns x if x is greater than or equal to zero
returns -x if x is less than zero

acos

Purpose: Computes the inverse cosine of a numeric quantity.

Form: *acos*(x)

Argument: any numeric quantity in the range -1 to 1

Return value: The return value is the angle whose cosine is equal to the argument, expressed in radians.

Return type: real

Examples: **acos(0.5)** returns 1.0472 radians (60 degrees)

See Also: *asin, atan, atan2, cos*

asin

Purpose: Compute the inverse sine of a numeric quantity.

Form: *asin*(x)

Argument: any numeric quantity in the range -1 to 1

Return value: The return value is the angle whose sine is equal to the argument, expressed in radians.

Return type: real

Examples: **asin(0.5)** returns 0.523599 radians (30 degrees)

See Also: *acos, atan, atan2, sin*

atan

Purpose: Compute the inverse tangent of a numeric quantity.

Form: *atan*(x)

Argument: any numeric quantity

Return value: The return value is the angle whose tangent is equal to the argument, expressed in radians.

Return type: real

Examples: **atan(100)** returns 1.5608 radians (89.5 degrees)

See Also: *acos, asin, atan2, tan*

atan2

Purpose: Compute the inverse tangent of a ratio. This is similar to atan(x/y), except that it preserves the individual signs of the arguments. This allows atan2 to return values over a full 2π radians.

Form: *atan2*(x, y)

Arguments: any numeric quantities

Return value: The return value is the angle whose tangent is equal to the ratio of the arguments, expressed in radians.

Return type: real

Examples:
atan2(100, 100) returns 0.785398 radians (45 degrees)
atan2(-1, -1) returns -2.35619 radians (-135 degrees)

Note: atan2(0, 0) is undefined.

See Also: *acos, asin, atan, tan*

ceil

Purpose: Returns the ceiling of a numeric quantity.

Form: *ceil*(x)

Argument: any numeric quantity

Return value: the smallest integer >= x

Return type: integer

Examples:
ceil(1.1) returns 2
ceil(-11.3) returns -11

See Also: *floor, round, trunc*

chr

Purpose:	Converts a numeric value to a character.
Form:	*chr*(x)
Argument:	numeric value in the range 1 to 255
Return value:	string containing the character with value equal to the argument
Return type:	string
Examples	**#H41** returns the character 'A'

cond

Purpose: Tests a status condition after a wait instruction. When a *wait for any status* statement is executed, this function can be called to determine which enabled status conditions caused the wait to terminate.

Form: *cond*(x)

Argument: status condition string (see *status* statement) or 'TIMEOUT'.

Return value: Zero is returned if the given condition was not set. One is returned if the condition was set.

Return type: integer

Examples:
```
cond('PROCESSING DONE 1')
cond('STORE DONE')
cond('TIMEOUT')
```

See Also: *status* and *wait* statements

cos

Purpose: Computes the cosine of an angle expressed in radians.

Form: *cos*(x)

Argument: a numeric quantity representing an angle expressed in radians

Return value: the cosine of the angle

Return type: real

Examples:
```
x=60
cos(x*pi/180)      returns 0.5
```

See Also: *acos, sin, tan*

exp

Purpose: Computes the value of e raised to the power of the argument.

Form: *exp*(x)

Argument: any numeric quantity

Return value: e ^ x

Return type: real

Examples:
t=3.5
exp(-2 * t) returns 0.000911882

Note: e is the base of the natural logarithm, 2.718281828...

See Also: *log, log10*

first

Purpose:	Returns the first element in a list.
Form:	*first*(list_name)
Argument:	the name of a list previously defined in the program
Return value:	the first element of the list
Return type:	element of the indicated list
Examples:	**list colors is (red orange yellow green blue violet)** **first(colors)** returns red
See Also:	*last, next, prev, list* statement

floor

Purpose: Returns the floor of a numeric quantity.

Form: *floor*(x)

Argument: any numeric quantity

Return value: the largest integer <= x

Return type: integer

Examples:
floor(1.1) returns 1
floor(-11.3) returns -12

See Also: *ceil, round, trunc*

format

Purpose: *Format* a string. This function is similar to the sprintf function in the language C.

Form: *format*(control, arg1, arg2, ..., argN)

Arguments:

control This argument is a control string which specifies how the remaining arguments are to be interpreted. The control string consists of types of objects: normal characters and conversion specifications. Normal characters are copied into the resulting string. Conversion specifications indicate how the remaining arguments are to be formatted.

A conversion specification begins with a percent (%) and ends with a character which indicates the type of conversion. The conversion characters are:

character	meaning
c	The corresponding argument is treated as an integer. Its value is used to produce an ASCII character. For example, if the value is 65, the letter "A" results.
d	The corresponding argument is treated as an integer. Its value is converted as a signed decimal number.
e	The corresponding argument is treated as a real. The number is converted in scientific notation, as in 1.234e-10.
f	The corresponding argument is treated as a real. The number is converted without an exponent, as in -123.456.
g	The corresponding argument is treated as a real. The number is converted using either the "e" or "f" form above, depending on the value, in order to reduce the number of characters in the result.
o	The corresponding argument is treated as an integer. Its value is converted as an unsigned octal number.

format (continued)

character	meaning
s	The corresponding argument is treated as a string. The text of the argument is copied into the result.
u	The corresponding argument is treated as an integer. Its value is converted as an unsigned decimal number.
x	The corresponding argument is treated as an integer. Its value is converted as an unsigned hexadecimal number. The digits 10 to 15 are represented by the letters "a" to "f".
X	The corresponding argument is treated as an integer. Its value is converted as an unsigned hexadecimal number. The digits 10 to 15 are represented by the letters "A" to "F".

You can further control the formatting by including a control number between the % and the conversion character.

The integer part of the control number indicates the minimum number of characters to be used to print the argument. '%6d' means that the argument is to be printed as a 6 digit integer. If the value doesn't require 6 digits, it is preceded by enough spaces to make 6 characters.

If the control number begins with zero, the value is preceded by zeros rather than spaces in order to fill the required number of characters. If '%06d' were used to format the value 123, the result would be 000123.

If the control number is negative, the value is left justified in the number of characters specified. If '%-6d' were used to format the value 123, the result would be '123 '.

When formatting real arguments, you may specify the number of digits to the right of the decimal point by including a fractional part in the control number. For example, '%6.3f' means that a real argument is to be formatted using a total of 6 characters, with three

format (continued)

to the right of the decimal point. Using this control with the value 1.23 would produce ' 1.230'.

arg1... additional arguments as required. These arguments must match their corresponding entries in the control string.

Return value: the formatted string

Return type: string

Examples: **format('value = %4.2f', 3.1)** returns 'value = 3.10'

Note: At most 10 arguments, including the control string, can be used.

inp

Purpose: Inputs value from an I/O port.

Form: *inp*(addr, size)

Arguments: addr - a numeric quantity specifying the I/O port address
size - the size of the I/O port in bytes (1, 2, or 4)

Return value: the value read from the I/O port.

Return type: integer

Examples: **inp(0x21,1)** returns the current interrupt mask value

See Also: *outp*

last

Purpose: Returns the last element in a list.

Form: *last*(list_name)

Argument: the name of a list previously defined in the program

Return value: the last element of the list

Return type: element of the indicated list

Examples:
list colors is (red orange yellow green blue violet)
last(colors) returns violet

See Also: *first, next, prev, list* statement

left

Purpose: Returns a substring containing the leftmost n characters of a string.

Form: *left*(x, n)

Arguments: x — a string
n — a numeric quantity

Return value: the leftmost n characters of the string x

Return type: string

Examples: **left('This is a string', 4)** returns 'This'

See Also: *mid, right, search, token*

log

Purpose: Returns the natural logarithm of a value.

Form: *log*(x)

Argument: a numeric quantity greater than zero

Return value: the natural logarithm of the argument

Return type: real

Examples:
log(1) returns 0
log(10) returns 2.30259

See Also: *exp, log10*

log10

Purpose: Returns the base 10 logarithm of a value.

Form: *log10*(x)

Argument: a numeric quantity greater than zero

Return value: the base 10 logarithm of the argument

Return type: real

Examples:
log10(1) returns 0
log10(10) returns 1

See Also: *exp*, *log*

mid

Purpose: Returns a substring of a string.

Form: *mid*(x, f, n)

Argument:
- x a string
- f a numeric quantity indicating the position of the first character of the substring. Zero indicates the first character of the string.
- n a numeric quantity indicating the number of characters in the substring.

Return value: the substring of the string x

Return type: string

Examples: **mid('This is a string', 5, 2)** returns 'is'

See Also: *left, right, search, token*

next

Purpose: Returns the next element in a list.

Form: *next*(x)

Argument: an element of any list defined in the program

Return value: the next element of the list

Return type: same as the argument

Examples:

list colors is (red orange yellow green blue violet)
c = first(colors)
next(c) returns orange

Note: If the argument is the last element of its list, the first element is returned.

The *next* function needs to know the list to which the argument belongs. Usually, it is clear. In the above example, the variable "c" belongs to the list "colors", because it was set to *first* (colors).

If an element appears in more than one list, ambiguity may result. In the following example, the values 0, 10, and 20 are common to two lists.

```
list multiples_of_five is (0 5 10 15 20)
list multiples_of_two is (0 2 4 6 8 10 12 14 16 18 20)
a = first(multiples_of_five)
b = 0
c = 2
x = next(a)
y = next(b)
z = next(c)
```

The first two assignments set the variables "a" and "b" to the value 0. Since "a" is clearly from the list "multiplies_of_five", the value of "x" must be 5. But "b" could belong to either list, so the value of "y" is not well defined. The third assignment sets "c" to the value 2. Since only the list "multiples_of_two" contains that value, "z" will be set to 4.

See Also: *first, last, prev, list* statement

ord

Purpose: Returns the ordinal number of a list element in a list.

Form: *ord(1, x)*

Arguments:

list_name	the name of any list defined in the program
x	any element of the list

Return value: The number of the element in the list. The first element of a list gives zero.

Return type: integer

Examples:

```
list colors is (red orange yellow green blue violet)
ord (colors, green)        returns 3

list dir is (north east south west)
array real data [ord(dir, north):ord(dir, west)]
```

This example uses the *ord* function to define the range of subscripts for an array. In this case, the array contains one element for each entry in the list dir.

Notes: If the element does not belong to the list, the program is terminated with an error message.

In contrast to the *next* and *prev* functions, *ord* requires that the name of the list be specified. This avoids ambiguities when the list element belongs to more than one list.

See Also: *array* and *list* statements
built-in functions *first*, *last*, *next*, and *prev*

outp

Purpose: Outputs a value to an I/O port.

Form: *outp*(addr, value, size)

Arguments: addr - a numeric quantity specifying the I/O port address
value - a numeric quantity to be written to the I/O port
size - the size of the I/O port in bytes (1, 2, or 4)

Return Value: This function does not return a value. It is used as a statement rather than inside an expression.

Examples: **outp(0x21, 0xff, 1)** sets the interrupt mask to all ones

See Also: *inp*

prev

Purpose: Returns the previous element in a list.

Form: *prev*(x)

Argument: an element of any list defined in the program

Return value: the previous element of the list

Return type: same as the argument

Examples:

list colors is (red orange yellow green blue violet)
c = last(colors)
prev(c) returns blue

Note: If the argument is the first element of its list, the last element is returned.

The *prev* function needs to know the list to which the argument belongs. Usually, it is clear. In the above example, the variable "c" belongs to the list "colors", because it was set to *last*(colors).

If an element appears in more than one list, ambiguity may result. In the following example, the values 0, 10, and 20 are common to two lists.

```
list multiples_of_five is (0 5 10 15 20)
list multiples_of_two is (0 2 4 6 8 10 12 14 16 18 20)
a = last(multiples_of_five)
b = 20
c = 2
x = prev(a)
y = prev(b)
z = prev(c)
```

The first two assignments set the variables "a" and "b" to the value 20. Since "a" is clearly from the list "multiplies_of_five", the value of "x" must be 15. But "b" could belong to either list, so the value of "y" is not well defined. The third assignment sets "c" to the value 2. Since only the list "multiples_of_two" contains that value, "z" will be set to 0.

See Also: *last, next, prev, list* statement

query

Purpose: Query system information. This function executes a normal GPIB query and returns the result as a string.

Form: *query*(x)

Argument: GPIB query string

Return value: result of the query

Return type: string

Examples: **query('GRID?')** returns 'GRID DUAL' if two grids are displayed

The following example prints trace definitions to the host.

```
list traces is (T1 T2 T3 T4 T5 T6 T7 T8)
for t in traces
   println host query(t|':DEF?')
endloop
```

Notes: The *query* function asks the system not to return a heading in the resulting string.

The *query* function should only be used when the result is a string. Do not use it when binary data is returned. For example, query ('WF?') is not allowed.

See Also: *left, mid, right, search, token, type*

right

Purpose:	Returns a substring containing the rightmost n characters of a string.
Form:	*right*(x, n)
Arguments:	x — a string n — a numeric quantity
Return value:	the rightmost n characters of the string x
Return type:	string
Examples:	**right('This is a string', 6)** returns 'string'
See Also:	*left, mid, search, token, type*

round

Purpose: Rounds a real value to the nearest integer.

Form: *round*(x)

Argument: a numeric quantity

Return value: the integer nearest to x

Return type: integer

Examples:

```
round(1.7)      returns 2
round(-100.1)   returns -100
```

See Also: *ceil, floor, trunc*

search

Purpose: Searches a string for a substring.

Form: *search*(x, s)

Arguments:
x — the string to be searched
s — the substring to be searched for

Return value: The position in string x of the first character of the substring s. The first position is zero. If the substring is not found, -1 is returned.

Return type: integer

Examples:

search('This is a string', 'is')	returns 2
search('This is a string', 'IS')	returns -1

sign

Purpose:	Return the sign of a numeric quantity.
Form:	*sign*(x)
Argument:	any numeric value
Return value:	Return 1 if the argument is >0, 0 if the argument is = 0, and -1 if the argument is < 0.
Return type:	integer
Examples:	**sign(x)**
Note:	The *sign* function is not related to the *sin* function.

sin

Purpose: Computes the sine of an angle expressed in radians.

Form: *sin*(x)

Argument: a numeric quantity representing an angle expressed in radians

Return value: the sine of the angle

Return type: real

Examples:
```
x=30
sin(x*pi/180)      returns 0.5
```

See Also: *asin, cos, tan*

sqrt

Purpose: Returns the square root of a numeric quantity.

Form: *sqrt*(x)

Argument: a non-negative numeric value

Return value: the square root of the argument

Return type: real

Examples: **sqrt(2)** returns 1.41421

strlen

Purpose:	Returns the length of a string.
Form:	*strlen*(x)
Argument:	a string
Return value:	the number of characters in the string argument
Return type:	integer
Examples:	**strlen('This is a string')** returns 16

tan

Purpose: Computes the tangent of an angle expressed in radians.

Form: *tan*(x)

Argument: a numeric quantity representing an angle expressed in radians

Return value: the tangent of the angle

Return type: real

Examples:
```
x=45
tan(x*pi/180)      returns 1
```

See Also: *atan, cos, sin*

token

Purpose:

Returns a token from a string. This function is useful in breaking a string down into its components. Often, you won't know exactly where each part of the string begins. This makes it difficult to use the *left, mid,* and *right* string functions. *Token* allows you to extract a substring from a string simply by knowing how many items precede it.

There are three types of tokens: identifiers, numbers, and everything else. Identifiers follow the same rules as variable names. Numbers may be integer or real, and may include a leading "+" or "-" sign, provided that there is no whitespace between the sign and the number. Any character which does not begin an identifier or a number is considered to be a token by itself, with the exception that whitespace characters are ignored.

Form:

token(x, n)

Arguments:

x — the string to examine
n — the number of the token to return, beginning with 1

Return value:

The token substring, or 'NO_TOKEN' if the n-th token was not present. This will happen if n is less than or equal to zero, or greater than the number of the last token in the string.

Return type:

string

Examples:

```
for n = 1 to 100
  t = token('example <= 1 ; -1.23e5.', n)
  print display n || t
  if t = 'NO_TOKEN'
    break
  endif
endloop
```

token (continued)

The preceding example prints the following tokens:

```
1    example
2    <
3    =
4    1
5    ;
6    -1.23e5
7    .
8    NO_TOKEN
```

Note: When in doubt about the number of a *token* in a string, write a loop similar to the above example to print the tokens in it.

See Also: *left, mid, right, search, type*

trunc

Purpose: Truncates a real value to an integer.

Form: *trunc*(x)

Argument: a numeric quantity

Return value: The argument is truncated toward zero. If x is greater than zero, the result is the same as *floor*(x). If x is less than zero, the result is the same as *ceil*(x).

Return type: integer

Examples:

trunc(1.7)	returns 1
trunc(-100.1)	returns -100

Note: In most other languages, this operation happens automatically whenever a real number is assigned to an integer.

See Also: *ceil, floor, round*

type

Purpose: Returns *type* information about an expression.

Form: *type*(x)

Argument: an expression (typically a variable name)

Return value:
0 if the expression is an integer
1 if the expression is a real
2 if the expression is an undefined variable
3 all other cases (string or list constant)

Return type: integer

Examples: **type(variable_name)**

Note: When a program is first executed, all variables are marked as undefined. The *type* function can be used to determine whether a value has ever been assigned to a variable, and if so, what type it is.

Since elements in a list need not all be of the same type, the *type* function can be used to determine the type of a particular list entry.

See Also: *list* statement

upper

Purpose:	Converts a string to upper case. Also, leading and trailing whitespace is removed, and each sequence of multiple whitespace characters is converted to a single space.
Form:	*upper*(x)
Argument:	a string quantity
Return value:	A string containing a "standard" version of the argument.
Return type:	string
Examples:	**upper(' This is a string ')** returns 'THIS IS A STRING'
Note:	This function is used internally in approximate comparisons of strings.

Section 8: GPIB Programming Examples

Following is a programming example in BASIC illustrating how to control GPIB input and output to the 7200. The program will run on an IBM PC with DOS 3.2 or higher and a National Instrument's GPIB-PC board installed. The file "bib.m" is the BASIC language interface file contained in the distribution diskette along with the GPIB-PC board. This binary file must be present in the same directory as the programming example in order to properly run the program. The program illustrates the use of status bytes in synchronizing remote control of the 7200. The program acquires waveforms and tests them against user-entered limits. The user also specifies whether acquired waveforms should meet or exceed these limits. Any waveform which meets these requirements are read over the GPIB bus and saved on disk in the current directory in files labeled "BABYSIT.000","BABYSIT.001", and so on up to "BABYSIT.999". This allows unattended monitoring of signals where only the waveforms of interest are saved. These waveforms may then be written back into the 7200 for later analysis. A brief line by line explanation follows:

LINE #	DESCRIPTION
60 -110	Initializes the GPIB handler. Lines 90 - 110 are taken from the file DECL.BAS. If these lines differ from your version, replace with your version of DECL.BAS.
120-240	Initialize the user limits, then loop forever acquiring waveforms, checking the pulse parameters against the user limits, and reading any waveforms satisfying the user limits which then get stored to disk.
250-740	Initialization Procedure:
	2020-2120: Initialize GPIB devices by sending Interface Clear, remote enable, and disabled timeout. Lastly, send "LOCAL" to allow simultaneous front-panel and remote operation for flexibility.
	300-310: Set the COMM_HEADER to short for faster transfers. Set the COMM_FORMAT to #9 definite length block with 16-bit data words in binary encoding for fast waveform transfers and easy read-back to the 7200. Set trigger mode to SINGLE. Turn on Trace 1.
	320-370: Turn off Trace 2 thru Trace 8.

380-510: Input the user-specified limits to monitor, set the default filename to "BABYSIT.000".

520-550: Define the user-specified Trace Equation for Trace 1.

560-700: Initialize internal variables based on user input.

710-740: Clear all status registers, enable the Waveform Processing Done mask for Trace 1, enable the Data Processing Done mask for Waveform Processing, and enable bit 1 of the Status Byte register. Thus, a service request (RQS) is only generated when processing is done for Trace 1.

750-800: Outside Procedure: If the Pulse Parameter is outside the user-specified limits, call the read/save procedure.

810-850: Inside Procedure: If the Pulse Parameter is inside the user-specified limits, call the read/save procedure.

860-1100: Acquire Procedure: Issues a trigger, acquires a waveform, and finds the user-specified pulse parameter.

860-890: Clear all status bytes and issue a trigger.

900-950: Poll for RQS for 20 seconds. If no RQS in 20 seconds, display a trigger timeout error and terminate. (TIMER returns the elapsed number of seconds).

960-990: If Waveform Processing complete, then query the Pulse Parameter else display error message: wrong bit set.

1000-1100: Read the Pulse Parameter and extract the 3rd token which is the value. If the value is "-", the pulse parameter is invalid else return the value. (Sample response is T1:PAVA FREQ, 5.135E6 Hz, AV if parameter is valid and T1:PAVA FREQ, - - -,IV if the parameter is invalid).

1110-1240: Read/Save Procedure: Sound the 7200 bell to alert the user that a waveform has exceeded the limits and query the waveform for Trace 1. Read the waveform into the disk file and increment the file extension. Check for errors, then print the number of bytes transferred.

1250-1600: Error Handlers: These subroutines handle any errors which may have occurred and set STATUS=1 which terminates the program. For a description of the variables IBSTA% and IBERR%, refer to the GPIB-PC User Manual under Status Word and Error Codes, respectively.

1610-2010: Token Parsing Subroutine: Parses a remote response string from the 7200 into individual tokens. The caller initializes TOKEN to the number of the token to be parsed, TOKENMSG$ is set to the response string, and TOKEN$ is the parsed token. For example, the query "T1:PAVA? FREQ" may return the response "T1:PAVA FREQ,20E+6 Hz,AV" where

TOKEN 1 = T1:PAVA	(header - Parameter value for Trace 1)
TOKEN 2 = FREQ	(keyword - Frequency)
TOKEN 3 = 20E+6	(value - 20MHz)
TOKEN 4 = Hz	(base units - Hertz)
TOKEN 5 = AV	(state - Averaged)

1610-1690: Parse from the first token to the requested token, return the requested token in TOKEN$

1700-1820: Step through the string character by character looking for either a comma,space,single quote, or double quote. Skip quoted strings and return tokens delimited by commas or spaces

1830-1900: Skip any leading or trailing spaces and return the token in TOKEN$

1910-1950: Skip single-quoted strings

1960-2010: Skip double-quoted strings

2130: Program END

Program Listing

```
10 REM  This program is used to acquire waveforms and read
20 REM  them over the GPIB to be saved on disk when a
30 REM  selected pulse parameter is between or outside of
40 REM  two user-specified values.
50 REM
60 REM  Load National Instrument's GPIB handler
70 REM  **NOTE: You must have the file "bib.m" in your current directory
80 REM  These lines merged from "DECL.BAS"
90 CLEAR ,60000! : IBINIT1=60000! : IBINIT2=IBINIT1+3 : BLOAD "bib.m",IBINIT1
100 CALL IBINIT1(IBFIND,IBTRG,IBCLR,IBPCT,IBSIC,IBLOC,IBPPC,IBBNA,IBONL,IBRSC,IB
SRE,IBRSV,IBPAD,IBSAD,IBIST,IBDMA,IBEOS,IBTMO,IBEOT,IBRDF,IBWRTF,IBTRAP)
110 CALL IBINIT2(IBGTS,IBCAC,IBWAIT,IBPOKE,IBWRT,IBWRTA,IBCMD,IBCMDA,IBRD,IBRDA,
IBSTOP,IBRPP,IBRSP,IBDIAG,IBXTRC,IBRDI,IBWRTI,IBRDIA,IBWRTIA,IBSTA%,IBERR%,IBCNT
%)
120  REM
```

```
130 GOSUB 250
140 REM

150 REM          ********* MAINLOOP *************
160 REM
170 STATUS=0
180 GOSUB 860
190 IF STATUS=1 THEN GOTO 2130
200 IF STATUS=2 THEN PRINT PULSE$+" Parameter is INVALID"
210 IF STATUS=0 THEN IF WHEN=1 THEN GOSUB 750 ELSE GOSUB 810
220 GOTO 150
230 REM
240 REM
250 REM          ******* INIT PROCEDURE *********
260 REM
270 INPUT "ENTER GPIB ADDRESS OF THE 7200 ", ADDR%
280 PRINT "INITIALIZING NATIONAL INSTRUMENT'S GPIB CARD"
290 GOSUB 2020
300 MSG$="CHDR SHORT;CFMT DEF9,WORD,BIN;TRMD SINGLE;T1:TRACE ON"
310 CALL IBWRT(DSO%,MSG$)
320 FOR T=2 TO 8
330   T$=STRING$(1,T+48)
340   MSG$="T"+T$+":TRACE OFF"
350   CALL IBWRT(DSO%, MSG$)
360 NEXT T
370 REM
380 INPUT "Channel to monitor (A1, A2, B1 or B2) ",CHAN$
390 INPUT "Parameter to monitor (1=frequency, 2=period, 3=peak-to-peak) ",PP
400 INPUT "Read waveform when parameter (1=outside, 2=inside) hi/lo limit ",WHEN
410 RANGE$="(0.00 to 1000.00) "
420 IF PP=1 THEN UNITS$="MHz "
430 IF PP=2 THEN UNITS$="uS "
440 IF PP=3 THEN UNITS$="Volts "
450 PRINT "HIGH LIMIT "+RANGE$+UNITS$;
460 INPUT HIGH
470 PRINT "Low limit  "+RANGE$+UNITS$;
480 INPUT LOW
490 FILEDIR$="BABYSIT"
500 FILEXT=1000
510 REM
520 QUOTE1$=CHR$(39):QUOTE2$=CHR$(34)
530 TRACE$="T1:DEF EQN,"+QUOTE2$+CHAN$+QUOTE2$+",MAXPTS,10000"
540 CALL IBWRT(DSO%, TRACE$)
550 REM
560 IF LOW < HIGH THEN GOTO 590
570    LOW = .9*HIGH
580    PRINT "High =";HIGH;" Low =";LOW
590 REM
600 IF PP=1 THEN MULT=1000000!
610 IF PP=2 THEN MULT=.000001
620 IF PP=3 THEN MULT=1
630 REM
```

```
640 IF PP=1 THEN PULSE$="FREQ"
650 IF PP=2 THEN PULSE$="PER"
660 IF PP=3 THEN PULSE$="PKPK"
670 REM
680 MAX = HIGH*MULT
690 MIN = LOW*MULT
700 REM
710 MSG$="*CLS;WPE 1;DPE 2;*SRE 2"
720 CALL IBWRT(DSO%, MSG$)
730 RETURN
740 REM
750 REM          ******** OUTSIDE PROCEDURE ***************
760 REM
770 IF VALUE < MIN THEN GOSUB 1110
780 IF VALUE > MAX THEN GOSUB 1110
790 RETURN
800 REM
810 REM          ******** INSIDE PROCEDURE ****************
820 REM
830 IF VALUE >= MIN THEN IF VALUE <= MAX THEN GOSUB 1110
840 RETURN
850 REM
860 REM          ******** ACQUIRE PROCEDURE ***************
870 REM
880 MSG$="*CLS;*TRG"
890 CALL IBWRT(DSO%, MSG$)
900 START=TIMER
910  REM Wait Loop
920    CALL IBRSP(DSO%, SPR%)
930    IF SPR% AND &H40 THEN GOTO 960
940    IF TIMER-START > 20 GOTO 1320
950    GOTO 910
960  REM Waveform Data Ready
970  IF NOT SPR% AND 2 THEN GOTO 1270
980  MSG$="T1:PAVA? "+PULSE$
990  CALL IBWRT(DSO%, MSG$)
1000 RESULT$=SPACE$(100)
1010 CALL IBRD(DSO%,RESULT$)
1020 TOKENMSG$=RESULT$:TOKEN=3:GOSUB 1630
1030 IF LEFT$(TOKEN$,1)<>"-" THEN GOTO 1060
1040   STATUS=2
1050   RETURN
1060 REM Valid Data
1070 VALUE=VAL(TOKEN$)
1080 RETURN
1090 REM
1100 REM
1110 REM          ********** READ/SAVE WAVEFORM PROCEDURE ************
1120 REM
1130 MSG$="BUZZ PULSE;T1:WF?"
1140 CALL IBWRT(DSO%, MSG$)
1150 FLNAME$=FILEDIR$+"."+RIGHT$(STR$(FILEXT),3)
1160 CALL IBRDF(DSO%,FLNAME$)
```

```
1170 FILEXT=FILEXT+1
1180 REM
1190 REM Store Operation Complete: Check For Errors
1200 IF IBSTA% AND &H8000 THEN GOTO 1420
1210 IF IBSTA% AND &H4000 THEN GOTO 1370
1220 PRINT IBCNT%;" BYTES TRANSFERED TO FILE ";FLNAME$
1230 RETURN
1240 REM
1250 REM          ********** ERROR HANDLERS **********
1260 REM
1270 REM Error Reading SRQ Register (SPR%)
1280 PRINT "SRQ = ";SPR%;"  BUT SHOULD BE 66"
1290 STATUS=1
1300 RETURN
1310 REM
1320 REM Timeout Error
1330 PRINT "TIMEOUT WAITING FOR TRIGGER"
1340 STATUS=1
1350 RETURN
1360 REM
1370 REM Timeout Error
1380 PRINT "TIMEOUT WRITING WAVEFORM TO DISK FILE"
1390 STATUS=1
1400 RETURN
1410 REM
1420 REM GPIB Error Condition: Interpret IBERR%
1430 REM
1440 IF IBERR% = 0  THEN PRINT "DOS ERROR"
1450 IF IBERR% = 1  THEN PRINT "FUNCTION REQUIRES GPIB-PC TO BE CIC"
1460 IF IBERR% = 2  THEN PRINT "NO LISTENER ON WRITE FUNCTION"
1470 IF IBERR% = 3  THEN PRINT "GPIB-PC NOT ADDRESSED CORRECTLY"
1480 IF IBERR% = 4  THEN PRINT "INVALID ARGUMENT TO FUNCTION CALL"
1490 IF IBERR% = 5  THEN PRINT "GPIB-PC NOT SYSTEM CONTROLLER AS REQUIRED"
1500 IF IBERR% = 6  THEN PRINT "I/O OPERATION ABORTED"
1510 IF IBERR% = 7  THEN PRINT "NON-EXISTENT GPIB-PC BOARD"
1520 IF IBERR% = 10 THEN PRINT "I/O STARTED BEFORE PREVIOUS OPERATION COMPLETED"
1530 IF IBERR% = 11 THEN PRINT "NO CAPABILITY FOR OPERATION"
1540 IF IBERR% = 12 THEN PRINT "FILE SYSTEM ERROR"
1550 IF IBERR% = 14 THEN PRINT "COMMAND ERROR DURING DEVICE CALL"
1560 IF IBERR% = 15 THEN PRINT "SERIAL POLL STATUS BYTE LOST"
1570 IF IBERR% = 16 THEN PRINT "SRQ STUCK IN ON POSITION"
1580 STATUS=1
1590 RETURN
1600 REM
1610 REM          ********** TOKEN PARSING PROCEDURE **********
1620 REM
1630 REM Token Parser: remove TOKEN token from TOKENMSG$, put in TOKEN$
1640   TOKINDEX=1
1650   TOKSIZE=LEN(TOKENMSG$)
1660   FOR TOKCOUNT=0 TO TOKEN-1
1670     IF TOKINDEX <= TOKSIZE THEN GOSUB 1700
1680   NEXT TOKCOUNT
1690 RETURN
```

```
1700 REM Return next token in TOKEN$
1710   TOKLENGTH=0
1720   TOK$=MID$(TOKENMSG$,TOKINDEX,1)
1730   IF TOK$="," THEN GOTO 1830
1740   IF TOK$=" " THEN GOTO 1830
1750   IF TOK$=QUOTE1$ THEN GOSUB 1910
1760   IF TOK$=QUOTE2$ THEN GOSUB 1960
1770   TOKLENGTH=TOKLENGTH+1
1780   TOKINDEX=TOKINDEX+1
1790   IF TOKINDEX <= TOKSIZE GOTO 1720
1800   IF TOKLENGTH=0 THEN TOKEN$=""
1810   IF TOKLENGTH>0 THEN TOKEN$=RIGHT$(LEFT$(TOKENMSG$,TOKINDEX-1),TOKLENGTH)
1820 RETURN
1830 REM Token Found
1840   IF TOKLENGTH=0 THEN TOKEN$=""
1850   IF TOKLENGTH>0 THEN TOKEN$=RIGHT$(LEFT$(TOKENMSG$,TOKINDEX-1),TOKLENGTH)
1860   TOKINDEX=TOKINDEX+1
1870 REM Skip leading/trailing white space
1880   IF TOKINDEX > TOKSIZE THEN RETURN
1890   IF MID$(TOKENMSG$,TOKINDEX,1)<>" " THEN RETURN
1900   TOKINDEX=TOKINDEX+1:GOTO 1870
1910 REM Skip to second closing single quote (')
1920   IF TOKINDEX = TOKSIZE THEN RETURN
1930   TOKLENGTH=TOKLENGTH+1
1940   TOKINDEX=TOKINDEX+1
1950   IF TOK$=QUOTE1$ THEN RETURN ELSE GOTO 1910
1960 REM Skip to second closing double quote (")
1970   IF TOKINDEX = TOKSIZE THEN RETURN
1980   TOKLENGTH=TOKLENGTH+1
1990   TOKINDEX=TOKINDEX+1
2000   TOK$=MID$(TOKENMSG$,TOKINDEX,1)
2010   IF TOK$=QUOTE2$ THEN RETURN ELSE GOTO 1960
2020 REM Subroutine : Initialize GPIB
2030 REM   Open devices & return unit descriptors
2040   DEVICE$="DEV"+CHR$(&H30+ADDR%)
2050   CALL IBFIND(DEVICE$,DSO%)                    ' Assign unit descriptor
2060   CNAME$="GPIB0":CALL IBFIND(CNAME$,BD%)
2070   CALL IBSIC(BD%)                              ' Send interface clear
2080   V%=1:CALL IBSRE(BD%,V%)                      ' Send remote enable
2090   V%=0:CALL IBTMO(DSO%,V%)                     ' Disable timeout
2100 MSG$="LOC"
2110 CALL IBWRT(DSO%,MSG$)                          'Enable local control
2120 RETURN
2130  END
```

APPENDIX A: SYSTEM MESSAGES

OPERATOR WARNINGS

An operator warning occurs when a command contains an illegal request but is automatically corrected by the 7200 (A). The command is completed using the correction. Some warnings will not cause any action to be taken but will simply display a warning message in the system message area.

CODE	MESSAGE	DESCRIPTION
1	"WARNING: DATA NOT ON A STEP"	Remote data was not a multiple of the step size, it was rounded to the nearest multiple.
2	"WARNING: DATA OUT OF RANGE"	Remote data was out of bounds; it was clipped to the nearest boundary.
3	"WAITING FOR HARDCOPY TO COMPLETE"	
4	"NO TRACES ARE ACTIVE"	Attempt to do something to a trace when none are active
5	"NO SETUP SCREEN FOR %s"	No menu available for this key (where %s is name of key)
6	"No help for key"	No help available for this key
7	"OUTPUT TO FLOPPY ABORTED" "OUTPUT TO FLOPPY ABORTED:code %d" "DISK IS FULL"	
8	"PLUGIN %c TIMEBASE TOO LOW FOR REALTIME"	Timebase too low for real time
9	reserved	A channel must be selected at all times
10	"PLUGIN %c - TIMEBASE TOO LOW FOR SEQUENCE"	Timebase too low for sequence mode
11	"PLUGIN %c RIS NOT PERMITTED IN SEQ MODE"	RIS not allowed during sequence mode
12	"PLUGIN %c TIMEBASE TOO HIGH FOR RIS"	Timebase too high for RIS
13	reserved	Mem size to low for realtime sequence mode.
14	"VERTICAL UNIT TOO LONG" "ILLEGAL VERTICAL UNIT" "HORIZONTAL UNIT TOO LONG" "ILLEGAL HORIZONTAL UNIT"	trace units to long too be displayed
15	"SEQ TRACE CANNOT BE RE-POSITIONED"	cannot change horizontal position of unexpanded seq. wfm

16	"WILL ABORT STORE AFTER FINISH CURRENT TRACE" "UNABLE TO STORE: INVALID TEMPLATE FILE" "NO TRACES TO BE STORED" "UNABLE TO RECALL: INVALID TEMPLATE FILE" "NO RECALL ITEMS SPECIFIED" "CANNOT WRITE TO AUTO BUFFER FILE" "NOT ENOUGH MEMORY TO RETRIEVE WAVEFORM" "%s IS NOT A VALID WAVEFORM FILE"	error storing or recalling to the disk
17	"PLUGIN %c - AMPLITUDE TO LARGE"	Autosetup - Amplitude to large
18	"Plugin %c - DC Offset too large to center signal"	Autosetup - DC offset too large to center trace
19	"Plugin %c - DC Offset too large to show signal"	Autosetup - DC component too large to show signal
20	"Plugin %c - DC Offset too large to center trace"	Autosetup - DC component too large to center trace
21	"Plugin %c - Amplitude too large to show signal"	Autosetup - Amplitude too large to show signal
22	not used	
23	not used	
24	not used	
25	"UNKNOWN FUNCTION %s" "UNKNOWN FUNCTION(CRT) %d"	unknown equation source in trace equation
26	"WARNING: NO PROGRAM DATA TO PRINT"	No data to print/plot to hardcopy device
27	"CANNOT OPEN TEMPLATE FILE" "CANNOT CLOSE TEMPLATE FILE" "CANNOT READ TEMPLATE FILE"	Error initializing template translation table
28	not used	
29	"DISPLAY OF WAVEFORMS DISABLED"	waveform display disabled
30	"KEY NOT ACTIVE"	key is not defined on current menu
31	"CANNOT EXECUTE FUNCTION WITH PLUG-INS LOCKED"	plugins timebase and or triggers are locked and control is not available since its not in common between the plug-ins.
32	not used	
33	not used	
34	not used	
35	"UNABLE TO COMPUTE CURSOR VALUE"	cannot compute cursor value for requested query

36	"CANNOT TURN TRACE OFF" "CANNOT TURN TRACE ON" "Incompatible traces" "Incompatible sampling interval" "Unallowed record type" "Non Overlapping time domains" "Needs 2 traces" "Invalid Traces"	trace cannot be turned on (in menu w/o traces or tre)
37	HF Sync cannot be used with LINE trigger	
38	"LIMIT REACHED"	no more values selectable by turning knob in same
39	"MUST USE "Edit Text" SOFTKEY TO EDIT"	user tried to edit text string without using "edit text" softkey
40	"CAN NOT COMPUTE PARAMETERS ON COMPLEX WAVEFORM" "CAN NOT COMPUTE PARAMETERS ON EXTREMA WAVEFORM"	can not calculate pulse parameters on this waveform type
41	INR Status bytes not set NO or Slow triggers	
42	"DOS FILENAMES CANNOT BEGIN WITH A NUMBER"	filename for hardcopy begins with a number
43	"PLOT ABORTED" "PRINT ABORTED"	Plot or print aborted
44	"NOT %s: ABORT IGNORED" "%s: SEND 'HCPY ABORT' TO ABORT" "In Local, Remote & Plot can't be both on GPIB" "Only %d percent of free plot memory" "HC:MEMORY ALLOC ERROR FOR GRAPHICS LINE" "Plot truncated, reduce either density or size" "Only %d percent of free plot memory" "Dump Device Slow, Disconnected or Hang"	Error trying to abort or during plotting or printing (where %s is printing or plotting)
45	"WARNING: END IN (PMT) SET TO 1" "WARNING: END IN (PMT) SET TO 127" "WARNING: LINE LENGTH SET TO 40" "WARNING: LINE LENGTH SET TO 1024"	Value adapted: argument to command adjusted to be valid
46	"PLUGIN %c - HF SYNC ACTIVE CANNOT MODIFY SLOPE"	Cannot modify slope when hf sync is active
47	"MISMATCH PROBE ATT CAN'T SELECT ALL CHANNELS"	Cannot lock vertical channel controls when Probe attenuations aren't equal
48	"MISMATCH PROBE ATT SELECTING CHANNEL %s"	Mismatch probe attenuation selecting channel
49	"PLUGIN %c - NO or SLOW EXTERNAL CLOCK"	External clock rate warning
50	"KEY LOCKED OUT DURING AUTOSETUP"	Key locked out during autosetup
51	"CANNOT CHANGE NUMBER OF GRIDS"	cannot change number of grids

52	"CAN'T MODIFY TRIGGER CONTROLS ON SLAVE" "KEY LOCKED OUT DURING AUTOSETUP"	cannot modify trigger controls on slave if plugins
53	"CANNOT OPEN FILE '%s'." "CANNOT REMOVE OLD FILE '%s'."	problems reading/writing floppy
54	"KEY LOCKED OUT DURING AUTOSETUP"	can't change plugin controls or timebase mode while
55	not used	
56	"PLUGIN %c CAN'T MODIFY SOURCE IN TV TRIGGER" "PLUGIN %c CAN'T MODIFY SLOPE IN TV TRIGGER" "PLUGIN %c CAN'T MODIFY COUPLING IN TV TRIGGER" "PLUGIN %c CAN'T MODIFY LEVEL IN TV TRIGGER"	Can't modify control while in tv trigger
57	"CANNOT QUERY ALL, CHANNELS NOT EQUAL"	Querying CHANNEL ALL when both channels do not have the same value
58	"FILE NOT PRESENT"	persistence file not found
59	"INTERNAL SYSTEM FAULT OCCURRED"	system error had occured
60	"PARAMETERS NOT ALLOWED IN PERSISTENCE OR XY"	cannot turn on parameters in xy or persistence
61	"WARNING: INACCURATE TDC."	TDC calibration error with external clock
62	"%c: NON-STANDARD EXTERNAL CLOCK FREQUENCY" "NON-STANDARD REFERENCE CLOCK FREQUENCY"	Dynamic calibration error with external clock
63	"ARM COMMAND TIMEOUT" "WAIT COMMAND TIMEOUT"	Wait command timeout
64	"OVERLOAD ON CH%d CAN'T SELECT ALL CHANNELS"	Overload is present, Can't select channel all
65	"PLUGIN %c CAN'T MODIFY SLOPE IN PATTERN TRIGGER"	Can't modify slope during pattern trigger
66	"PLUGIN %c CAN'T MODIFY HF SYNC IN SMART TRIGGER"	Can't modify hf sync during smart trigger
67	"PLUGIN %c CAN'T MODIFY VAR. V/DIV WITH 7291"	Variable volts/div can't be used with 7291 plugin
68	"PLUGIN %c CAN'T MODIFY VERT COUPLING WITH 7291"	Vertical input coupling can't be used with 7291 plugin
69	"POSITION LIMITED BY NON-DISPLAYED TRACE"	With common expand on, cannot horizontally reposition the traces any more in current direction due to a trace which is "locked" to the displayed group but is not being displayed.

70	"PLUGIN %c CAN'T MODIFY SLOPE IN LINE TRIGGER" "PLUGIN %c CAN'T MODIFY COUPLING IN LINE TRIGGER" "PLUGIN %c CAN'T MODIFY HF SYNC IN LINE TRIGGER" "PLUGIN %c CAN'T MODIFY LEVEL IN LINE TRIGGER"	Line triggers control settings are not used
71	"PLUGIN %c REVISION DATA INVALID"	EEPROM checksum failed
72	"PLUG-IN %c INVALID REVISION"	Module revision is not valid
73	"PLUGIN %c CAN'T MODIFY TRIG SOURCE WITH 7291"	Trigger source must always be CH1 with 7291 plugin
74	"SEGMENTS NOT LOCKED"	When traces are set for common horizontal expand & in sequence mode, they must have the same number of segments.
75	"PLUGIN %c RIS NOT VALID WITH 7291"	
76	"PLUGIN %c INVALID MEM LENGTH FOR THIS TIME PER DIV"	
77	"GPIB ERROR: UNSUPPORTED CMD: %d"	
78	"GPIB ERROR: NO LISTENERS ON BUS"	
79	"GPIB ERROR: CAN'T SEE 9914 GPIA"	
80	"GPIB ERROR: INVALID CONVERSION TYPE"	
81	"GPIB ERROR: OUTPUT TIMEOUT"	
82	"CENTRONICS ERROR: UNSUPPORTED CMD: %d"	
83	"CENTRONICS ERROR: OUT OF PAPER"	
84	"CENTRONICS ERROR: DEVICE NOT SELECTED"	
85	"CENTRONICS ERROR: TIMEOUT"	
86	"CENTRONICS ERROR: OUT OF PAPER"	
87	"CENTRONICS ERROR: PRINTER ERROR"	
88	"RS232 ERROR: UNSUPPORTED CMD: %d"	
89	"RS232 ERROR: PARITY ERROR"	
90	"RS232 ERROR: FRAMING ERROR"	
91	"RS232 ERROR: OVERRUN ERROR"	
92	"RS232 ERROR: BREAK RECEIVED"	
93	"RS232 ERROR: OUTPUT TIMEOUT"	
94	"GENERAL ERROR: ILLEGAL CMD: %d"	
95	"GENERAL ERROR: SOFTWARE ERROR"	
96	"GENERAL ERROR: UNKNOWN HARDCOPY DEVICE"	
97	"GENERAL ERROR: OUTPUT TIMEOUT"	

98	"GENERAL ERROR: OUT OF PAPER"	
99	"GENERAL ERROR: PRINTER ERROR"	
100	"UNKNOWN ERROR #%d"	
101	"TRYING TO LOCK DIFFERENT PLUGINS"	
102	"TRFI: ERROR READING FILE"	Problems writing/reading Floppy Disk for hardcopy
103	"PLUGIN %c CAN'T TRIGGER ON CAL SIGNAL"	
104	"PLUGIN %c CH1 CAL DATA NOT VALID"	
105	"PLUGIN %c CH2 CAL DATA NOT VALID"	
106	"PLUGIN %c CHANNEL MISMATCH"	
107	"WARNING: CHECK SEATING OF MODULE %c"	
108	"VALUE LIMITED BY LOCKED PLUG-IN"	A control tried to change on one plugin but because the corresponding control on the other locked plugin(s) plugin was set to the value dictated by the most limiting plugin.
109	"TRACE n: INPUT DATA WAS ZERO FILLED" "TRACE n: INPUT DATA HAS OVER/UNDERFLOWS" "TRACE n: NO INPUT DATA" "TRACE n: FFT INPUT INVALID" "TRACE n: FFT COMPLEX DATA INVALID" "TRACE n: PARAMETER IS NOT APPLICABLE" "TRACE n: PARAMETER CANNOT BE COMPUTED" "TRACE n: WAVEFORM HAS UNDER/OVERFLOWS" "TRACE n: WAVEFORM HAS UNDEFINED POINTS" "TRACE n: CANNOT PROCESS STRING PARAMETER"	
110	"CAN'T DISPLAY TRACE WHILE ROLLING"	This message is displayed when in persistence while traces are rolling
111	"PLUGIN n LEVEL IS DISABLED IN AUTO ROLL MODE"	This message is displayed when level is changed while in auto trigger mode
112	"PLUGIN n DOES NOT SUPPORT PROTECTED MODE"	module does not contain the BRAM needed to protect data
113	"PLUGIN n NO operation IN PROTECTED MODE"	module can not protect RIS, ROLL MODE, SYNC AVG, and PACKED SEQUENCE
114	"CANNOT ALLOCATE MEMORY FOR CHANNEL"	there is not enough mainframe memory for long memory support (7200A only)

115-149	reserved	
150	"FLOPPY DISK IS WRITE-PROTECTED"	
151	"FLOPPY DISK IS WRITE-PROTECTED OR FORMAT FAILED"	
152	"FLOPPY DISK IS NOT MSDOS FORMATTED"	
153	"FLOPPY DISK IS FULL"	
154	"CANNOT DELETE FILE"	
155	"CANNOT CREATE FILE"	
156	"CANNOT OPEN FILE"	
157	"CANNOT CLOSE FILE"	
158	"CANNOT MOUNT FLOPPY DISK"	
159	"ERROR FORMATTING FLOPPY"	
160	"ERROR WRITING TO FLOPPY DISK"	
161	"ERROR READING FROM FLOPPY DISK"	
162	"NO FLOPPY PRESENT IN DISK DRIVE"	
163	"FLOPPY BEING ACCESSED, FORMAT ABORTED"	
164	"CANNOT WRITE TO HARD DISK"	
165	"CANNOT READ HARD DISK"	
166	"CANNOT REMOVE OLD FILE"	
167	"CANNOT READ DIRECTORY"	
168	"CANNOT FIND FILE"	
169	"NOT ENOUGH MEMORY"	
170	"RECALL STILL IN PROGRESS"	
171	"STORE STILL IN PROGRESS"	
172	"INTERNAL DISK IS FULL"	
173	"INVALID FILE TYPE, COPY ABORTED"	

OPERATOR ERRORS

An operator error occurs when a command cannot be executed because it contains an illegal request. an error message is displayed and the appropriate codes are set in the Execution Error Register. The command is ignored.

CODE	MESSAGE	DESCRIPTION
200	"DATA ERROR: %s" "INVALID NUMBER BASE: #%s" "ILLEGAL CHARACTER: %c" "ERROR PARSING WF COMMAND" "TRFI: MISSING DISK PARAMETER" "TRFI: INVALID DISK PARAMETER" "TRFI: MISSING FILE PARAMETER" "TRFI: INVALID FILE PARAMETER" "TRFI: NO SUCH FILE" "ERROR PARSING STRING: %s" "EXTRA PARAMS IGNORED: %s" "ERROR PARSING CORS COMMAND" "ERROR: MISSING KEYWORDS AND VALUES" "ERROR: MISSING KEYWORD" "ERROR: UNKNOWN KEYWORD: %s" "ERROR: ILLEGAL KEYWORD: %s"	error parsing data (where %s is the data); the data for a remote command does not obey the syntax rules
201	not used	
202	not used	
203	"ILLEGAL PREFIX" "PLUG-IN NOT PRESENT"	illegal module ID in remote command or specified plugin is not present
204	"ILLEGAL CHANNEL"	illegal channel ID in remote command
205	not used	
206	"ILLEGAL UNITS"	bad units/multiplier in remote command
207	"UNKNOWN OPERAND: %s"	unknown enumerated type in remote command
208	"DATA IS NOT NUMERIC" "DATA NOT HEXADECIMAL" "DATA NOT OCTAL" "DATA NOT BINARY"	data from a remote (or RCL) command is not numeric or in the expected number base
209	"CNTRL NOT ALLOWED WITH PLUG-INS LOCKED"	tried to lock to a nonexistant key
210	"VALUE OUT OF RANGE WITH PLUG-INS LOCKED"	lock value out of range for other key(s)

211	"NO CHANNEL SPECIFIED"	remote command did not specify a channel or trace number
212	"NO CHANNELS AVAILABLE"	channel specified when it should not have been such as for a timebase or trigger command
213	"QUERY ON VERT CMDS MUST SPECIFY CHANNEL"	query function for vertical command recieved without a channel specified
214	not used	
215	"TRACE n: UNIT ERROR" "TRACE n: HORIZONTAL UNITS DON'T MATCH" "TRACE n: VERTICAL UNITS DON'T MATCH" "TRACE n: HORIZ RANGES DON'T MATCH" "TRACE n: DIVIDE BY ZERO" "TRACE n: LOG OF ZERO" "TRACE n: TOO MANY COEFFICIENTS" "TRACE n: INCOMPATIBLE RECORD TYPES" "TRACE n: HORIZ EXPAND ERROR" "TRACE n: INPUT HAS TOO FEW POINTS" "TRACE n: COMPUTATIONAL OVERFLOW" "TRACE n: OUT OF MEMORY" "TRACE n: PROCESSING ERROR" "TRACE n: CAN'T DECIMATE SEQUENCE WAVEFORM" "WAVEFORM PARAMETERS OUT OF MEMORY"	processing routine error
216	"NOT ON TOPIC HEADER"	Not on Topic Header
217	"ALREADY IN TABLE OF CONTENTS"	Already in TOC
218	"INVALID TRACE NUMBER"	invalid trace number for the trace equation sent over remote
219	not used	
220	"TRACE IS OFF, CANNOT SELECT"	cannot select trace which is turned off
221	not used	
222	not used	

223	"Error at line %d: %s" where %s is: "expr stack overflow" "floating point overflow" "division by zero" "second operand of mod is zero" "invalid operands of ^ operator" "undefined var" "string overflow" "number required" "var is loop index" "var waiting for input" "bad arg of acos fn" "bad arg of asin fn" "bad arg of log fn" "bad arg of log10 fn" "bad arg of sqrt fn" "Unsupported format option" "Too many arguments for format specified" "Too few arguments for format specified" "bad arg of next fn" "bad arg of prev fn" "unknown status" "wait with no status enabled" "proc stack overflow"	Auto Sequence Operator error (program errors) These messages are considered fatal execution errors as far as the user's program is concerned, which means that they abort any program in progress.

224	"PRSU: missing value" "PRSU: invalid disk '%s'." "PRSU: invalid file '%s'." "PRSU: invalid file '%s'." "PRSU: invalid speed '%s'." "PRSU: invalid keyword '%s'." "TURN: missing KNOB/POT parameter" "Can't insert keyword '%s'." "Incompatible symbols in data file." "Cannot open input file '%s'." "Cannot remove old file '%s'." "Cannot open output file '%s'." "Error reading from input file." "Error writing to output file." "Can't redefine '%s' as list constant." "Can't insert '%s'." "Incompatible strings in data file." "INVALID PANEL SETTINGS FILE\n" "Input file is empty." "Please recompile with version %ld of COMPILE." "\"SRC\" and \"APD\" files don't match." "%s: missing identifier" "%s: plugin not present" "PRSU: invalid keyword '%s'."	Auto Sequence Operator error (disk related errors) These messages are considered fatal operator errors as far as the user's program is concerned, which means that they abort any program in progress and usually (but not always) delete it from memory.
225	"INVALID KEYWORD IN COMMAND"	unknown keyword in remote command
226	"DUPLICATE TRACE STORAGE DESTINATION TRACES" "FILENAME MUST HAVE 3-DIGIT NUMERIC EXTENSION" "CANNOT TURN ON RECORD TRACES WHEN IN REPLAY MODE" "CANNOT TURN OFF RECORD TRACES WHEN IN REPLAY MODE" "THERE ARE NO RECORDED TRACES" "REPLAY TRACES MODE IS NOT ON" "FILENAME MUST BE MAX OF 8 CHARS WITH NO EXTENSION"	invalid store/recall setup remote command
227	"MUST SPECIFY PARAMETER" "MISSING VALUE" "MUST SPECIFY LOCATION (1-20)"	invalid remote command
228	not used	
229	"INVALID TRACE EQN: %s" "ILLEGAL VALUE: %s" "CANNOT SET %s IN THIS EQUATION"	Invalid trace equation or trace parameter

230	"ILLEGAL TRACE LABEL: TOO MANY CHARACTERS" "ILLEGAL TRACE LABEL: TOO FEW CHARACTERS" "ILLEGAL TRACE LABEL: %s" "ILLEGAL CHARACTER IN TRACE LABEL: %c"	Invalid trace label - must be a DOS compatible filenae
231	"NO SPACE FOR %s"	Output response buffer overflow
232	"OM: QUERY AFTER INDEFINITE RESPONSE DISCARDED"	Query response (following #0 or OFF trace query) flushed
233	"ERROR #%d TRYING TO ABORT HARDCOPY"	Error during plotting or printing
234	"WAVEFORM READ QUERY ERROR"	Parsing Error: invalid arguments following query
237	"NO TRACES TO BE PRINTED"	Trying to print trace data and trace is not turned on
238	"DOS FILENAMES CANNOT EXCEED 8 CHARACTERS" "COMMENTS MUST BE 40 CHARACTERS OR LESS"	
239	"NO TRACE NUMBER SPECIFIED"	Remote mainframe command received that needs a trace #
240	"ILLEGAL SOURCE"	Invalid source specified in remote command
241	"ILLEGAL TRACE"	Invalid active trace specified in remote command
243	"INVALID COMMAND TERMINATION"	The input manager scaned all remaining data but did not encounter a proper terminator.
244	"SECURITY KEY NOT PRESENT"	Remote command requires the security key (dongle), but it is not present.
245		Command can't be executed while in protected mode
246	"DATA PROTECTED WITH VERSION n"	Data was saved in BRAM with a certain version of software. To recover the data, move module into a mainframe containing the same version.
247	"SCSI DEVICE NOT READY"	The SCSI device indicated by the command did not respond to a SCSI test ready command. This error can occur when entering protected mode. In this case, protected mode SCSI operations are inhibited.

COMMAND ERRORS

These are errors encountered while trying to parse (interpret) a remote command such as, invalid syntax, semantic. These errors set the Command Error status bit.

CODE	MESSAGE	DESCRIPTION
450	"COMMAND ERROR: %s"	Unknown command - invalid syntax
451	"ESCAPE COMMAND ERROR: <ESC>%c"	Invalid escape command
452	"UNKNOWN OPERAND: %s"	Unknown operand
453	"ILLEGAL DELIMETER(s)"	bad delimiters (, ; etc) in remote command
454	"INVALID DATA: %s"	invalid data in remote command

QUERY ERRORS

These are errors caused when a remote computer tries to query the 7200(A) when no data is present or data is unavailable. These errors cause the query error status bit to be set.

CODE	MESSAGE	DESCRIPTION
402	Interrupted Action (IEEE488.2 Sect 6.3.2.3)	
403	Unterminated Action (IEEE488.2 Sect 6.3.2.2)	
404	Query After Indefinite Response (IEEE488.2 Sect 6.5.7.5)	
405	Buffer Deadlock (ie., T1:WF?;T1:WF ALL,#9NNNN...) (IEEE488.2 Sect 6.3.1.7)	
406	"QUERY ERROR: RESPONSE OVERFLOW"	Query response too large ... buffer overflow

INTERNAL MESSAGES

These errors are used to indicate that an internal consistency check failed but that operation of the system can continue normally. These messages should rarely occur and only if the same message appears repeatedly, should you contact the local service representative or the factory for futher details.

CODE	MESSAGE	DESCRIPTION
500	"PCON GAVE EXEC < 2K BUT HAS MORE PENDING" "TRE: COULD NOT FIND TOKEN WITH THESE UNITS"	
501	"INVALID ERROR NUMBER"	Invalid error number(bad parameter passed by caller)
502	"Illegal ACQ MGR message"	Illegal Message to ACQ MGR
503	"Plugin %c's current state is ready" "Plugin %c's current state is armed" "Plugin %c's current state is test" "Plugin %c's current state is processing" "Plugin %c's current state is wait retry" "Plugin %c's current state is delay test"	Invalid state transiiton
504	"illegal LED, device %d, number %d"	illegal led number specifed by caller
505	"bin copy: %d points < %d pixels"	invalid number of points to be binned
506	"interpolation: partial bin=%d" "unable to interpolate points"	tried to interpolate when no points assigned
507	"bad compaction factor=%d"	illegal compaction factor
508	"REM PARSE ERR:INVALID CMD TABLE ENTRY"	rem parsing err: unknown value type in COM_ENTRY
509	"KEY PARSE ERR:INVALID CMD TABLE ENTRY"	key/pot parsing err: unknown value type in COM_ENTRY
510	"cannot make options list(bad value type)");	options list err: unknown value type in COM_ENTRY
511	"module does not exist"	unknown plugin type (see CPR_mod_decode())
512	"MEMORY ALLOC ERROR FOR PREV_VAL_LIST"	cannot allocate memory for PREV_VAL_LIST
513	"MENU STACK OVERFLOW"	menu stack overflow
514	"MENU STACK UNDERFLOW"	menu stack underflow
515	COM_ENTRY's menu_index=0-7(TRACE) but plugin is not MF	

516	"MEMORY ALLOC ERROR FOR TRE SYSTEM" "MEMORY ALLOC ERROR FOR PROCESS_tree"	memory allocation error for prev val of struct TREE_NODE
517	"all tree nodes used"	no free tree nodes available in free node list
518	"tried to free an illegal greek letter" "ran out of greek letters to allocate"	no more available greek letters in free list
519	"units too large: %s"	remote form of units string exceeds COM_ENTRY size
520	trace edit stack overflow	
521	trace edit stack underflow	
522	"unknown units: %d"	token form of units string is unrecognized
523	not used	

524	memory allocation error for previous value of TOKEN "DFM ERROR: PSOS_Suspended node not in comp_process" "DFM ERROR: Running node not in comp_process" "DFM ERROR: COMP PROCESS ARRAY CORRUPTED" "DFM DID NOT RECEIVE REPLAY TRACE EQNS" "DFM ERROR: BAD TRACE EQN" "DFM DID NOT RECEIVE NORMAL TRACE EQNS" "INTERNAL DFM ERROR: DATA CORRUPTED" "DFM ERROR: BAD TRACE EQN" "DFM ERROR: INVALID INIT COMMAND" "DFM ERROR: BAD COMP PROC ID" "DFM ERROR: UNIMPLEMENTED COMMAND" "DFM ERROR: INVALID COMMAND" "DFM ERROR: INVALID WAVE POINTER TYPE TO FIXUP FN" "Fixup Function Called For Locked Buffer" "DFM INTERNAL ERROR #1" "DFM INTERNAL ERROR #2" "DFM INTERNAL ERROR #3" "DFM INTERNAL ERROR #4" "DFM INTERNAL ERROR #7" "DFM ERROR: INCONSISTENT PIP SETUP" "INTERNAL DFM ERROR: DATA CORRUPTED" "DFM ERROR: Ran out of spec structures" "DFM ERROR: Did not find parm spec" "DFM ERROR: Ran out of spec structures" "DFM ERROR: DIFFERENT INIT FOR PSOS SUSPENDED NODE" "DFM ERROR: GOT PLUGIN READY WHEN ALREADY HAVE CHANNEL" "DFM RECEIVED MEMORY READY WHEN ALREADY HAS IT" "COP ERROR: TRIED TO ALLOCATE SEQUENCE ARRAY"	internal data structures corrupted
	"COP ERROR: TRIED TO ALLOCATE SEQUENCE TRIG ARRAY" "COP ERROR: INVALID WAVE POINTER TYPE"	
525	"ILLEGAL MESSAGE SENT TO PIP %c"	Illegal msg received by PIP
526	"ERROR ATTACHING TO MSG XCHG")	Error attaching to ACQ MGR message exchange

527	"ERROR IDENITFYING DFM PROCESS"	Error identifying DFM process id
528	"ERROR IDENTIFYING DFM PROCESS"	Error identifying DFM process id
529		Invalid Led Id
530	"TRE: COULD NOT FIND COM_ENTRY FOR T%d"	cannot find COM_ENTRY in tables
531	"Can't open input file '%s'." "Can't allocate memory for category '%s'." "Can't allocate memory for category lists." "Can't allocate memory for unit lists." "Input error reading file '%s'." "Can't read unit heading line from file '%s'." "Can't read category heading line from file '%s'." "Can't allocate memory for unit table." "Can't allocate memory for string '%s'." "unknown"	general units error
531	"PLUGIN %c: MUST BE A REMOTE CMD"	
532	"TRE: INSUFFICIENT MEMORY TO SAVE TO BRAM" "TRE: INSUFFICIENT MEMORY TO RECALL FROM BRAM"	memory allocation error for struct TRE_SAVE_INFO
533	"MEMORY ALLOC ERROR FOR MENU SYSTEM" "MEMORY ALLOC ERROR FOR MEN_PREV_JR_MENU"	cannot allocate memory for MEN_PREV_JR_MENU error building display strings
534	"ERROR SENDING MSG TO DFM=%d"	Tried to send a message to DFM to read a trace/channel
535	"ERROR IN TRE—UNITS INTERFACE: CRT_LIST"	error creating list of units
536	"ERROR IN TRE—UNITS INTERFACE: REM_LIST"	too many intensified regions were found
537	"DSK SOFTWARE ERROR #2" "DSK SOFTWARE ERROR #3" "DSK SOFTWARE ERROR #4" "DSK ERROR: INVALID COMMAND" "DSK ERROR: RECEIVED CLEAR MEMORIES IN REPLAY TRACES" "DSK ERROR: INVALID WAVE POINTER TYPE TO FIXUP FN"	software error
538	Processing done interrupt error	
539	Internal PP error	
540	"Internal Program error %d"	Internal Auto Sequence error. The # indicates where in the code error occured. Report it to the software group.

541	"CHECKSUM ERROR, pHILE CODE CORRUPTED" "INSUFFICIENT MEMORY TO START pHILE" "pHILE THINKS VERIFY CODE IS BAD" "UNKNOWN pHILE ERROR DURING PHILE_INIT" "MOUNT_VOL: TOO MANY VOLUMES" "MOUNT_VOL: NOT A pHILE VOLUME" "MOUNT_VOL: VOLUME ALREADY MOUNTED" "MOUNT_VOL: ILLEGAL DEVICE" "MOUNT_VOL: ILLEGAL SYNCH MODE" "MOUNT_VOL: ILLEGAL DEVICE NAME" "MOUNT_VOL: UNKNOWN ERROR CODE"	error initializing pHILE
542	"DISK INITIALIZE ERROR"	Error initializing PCON disk operations
543	"Error opening %s" "Can't read HLP_INDEX_TABLE" "Can't alloc mem for prev_next array" "Can't read prev_next array" "Missing help data (.pna)"	Missing help data
544	"Can't alloc mem for HLP_INDEX_TABLE"	Memory allocation error
545	"PLUGIN %c CAN'T OPEN TMS FILE"	Error Opening TMS code
546	"PLUGIN %c CAN'T READ FROM FILE"	Error Reading TMS code
547	"PLUGIN %c TMS NOT RESET"	TMS not reset
548	Error reading Help file	
549	"DUMMY ROUTINE CALLED FROM MF_TBL"	no command processing routine specified for command
550	not used	
551	"ERROR: SPURIOUS INTERRUPT" or after a processor exception occurs, it is reported by one of the following abbreviations: "BE" "AE" "II" "ZD" "CHK" "TRP" "PV" "TR" "LE" "CPV" "FE" "UI" "SI" "L1AV" "L2AV" "L3AV" "L4AV" "L5AV" "L6AV" "L7AV" "T#0" "T#1" "T#2" "T#3" "T#4" "T#5" "T#6" "T#7" "T#8" "T#9" "T#10" "T#11" "T#12" "T#13" "T#14" "T#15" "FBUC" "FIR" "FZD" "FU" "FOE" "FO" "FSN" "MC" "MIO" "MALV" "IV%x" "UV%x" followed by: " PC %08lx" and then optionally: " IA %08lx", " DCFA %08lx", "Invalid Format"	spurious interrupt occurred (but ignored) or Exception (BE, AE, etc) occurred
552	TMS not responding during cal	
553	"*CAL COMMAND TIMEOUT"	*CAL command timeout
554	not used	
555	not used	
556	not used	
557	"ERROR DETERMINING MODULE TYPE - PLUGIN %c", Error determining module type	

558	not used	
559	"PLUGIN %c TMS NOT IDLE"	TMS not going idle as expected
560	"WARNING: TOO MANY JNR MENUS, MENU MAY NOT WORK"	Too many junior menus; possible irratic operation
561	"TRE: BRM BREAKDOWN(CODE 2)" "TRE: BRM BREAKDOWN" "TRE: UNKNOWN BRM_RECALL CODE: %d" "TRE: TRIED TO EXECUTE BU_RECALL WHILE ACTIVE"	Operation not valid/unknown action
562	"BRM: ILLEGAL ID" "BRM: BAD COUNT ON ID#%d" "BRM: INSUFFICIENT SPACE ON ID#%d\n"	Illegal or bad ID/no space on ID
563	"Error %d in Plot Preparation", PL_work.stat "HC ERR: MUST SELECT A PRINTER FOR OUTPUT" "HC ERR: UNKNOWN HC OUTPUT, HARDCOPY ABORTED"	Error during plotting or printing
564	"NO PCON RESPONSE TO ABORT COMMAND IN 1 SEC"	No PCON response to command
565	"CP ERROR: CAN'T READ BLOCK 2 OF HARD DISK"	Error reading Hard Disk
566	"Cannot set status bit %d"	Error trying to set status bits
567	"OM ERROR #%d KEY CMD %s" "PCON sent error #%d" "OM ERROR #%d RH MSG %s"	Error sending command to PCON
568	"OM: ERROR RECALLING IO PARAM FROM BRAM" "OM: UNKNOWN BRM_RECALL CODE: %d"	Error recalling from BRAM
569	"ERROR IDENTIFING TRM PROCESS"	Error identifying TRM process
570	Error identifying DSCH process	
571	Error during gain calibration	
572	"INVALID TRACE EQUATION FOR TRACE %d"	Error trying to create a tree node - invalid equation
573	"PLUGIN %c - ERROR SETTING FLASH MODE"	Error setting desired flash mode during calibration
574	"CAN'T ALLOCATE PERSISTENCE BUFFER" "CAN'T GET XY BUFFER"	Error allocating memory - Can't do persistence
575	"CANNOT DISPLAY TIMESTAMPS"	Error allocating memory - Can't display timestamps
576	"INSUFFICIENT MEMORY AVAILABLE FOR PLOT"	Error allocating memory for raster bit map
577	"PLUGIN %c ILLEGAL ACQUISITION DONE INTR"	ACQ DONE interrupt from non present plugin

578	Can't find current index for trig pattern when triggers are locked	
579	Can't abort COP processing Bad histogram/trend value received from DFM	
580	"UNKNOWN MANTISSA: %f" "UNKNOWN WIDTH: %dE%d"	The width field for a histogram is invalid
581	"TRIG TIME INIT TIMEOUT"	Trig time menu init timeout
582	"ERROR READING INPUT BLOCK DATA"	Requested input data from PCON but error occurred
583	"Error processing waveform data input"	Error processing waveform input data
584	"CAN'T %s PLUGIN DRIVER \"%s\"" "CAN'T LINK SYMBOL \"%s\"\nIN PLUGIN DRIVER \"%s\"" "CAN'T LOAD MODULE \"%s\""	Error loading plugin driver
585	EEPROM programed incorrectly	
586	"TRACE SETUP NOT AVAILABLE"	The com_entry for this softkey has unknown module ID
587	"INVALID TRACE EQUATION:T%d=%s"	The trace equation stored in BRAM is invalid
588	"TMS ERROR READING DATA MEMORY"	TMS has trouble loading tms program memory
589	"TRGSRC CHANNEL 4 NOT ALLOWED"	Channel 4 is not a valid trig source in P34
590	(7200A only) NMI interrupt	
591	"SCSI ABORT FAILURE"	An attempt was made to abort a SCSI transfer (exiting protected mode) but the hardware reported that the abort could not be performed. Operation continues normally, assuming that the transfer which was would have been aborted is not actually in progress.
592	"SCSI SETUP FAILURE"	An attempt was made to initiate a SCSI read or write operation, but the hardware reported that it was not possible. The error is ignored, but will probably result in further errors and incorrect operation of the SCSI hardware.

SYSTEM RESTART MESSAGES

These messages are reports when a hardware or software condition occurs that put the 7200(A) in an unrecoverable state and after the condition is logged the system is reset to its default state to try to clear the error. These messages should rarely occur and only if the same message appears repeatedly, should you contact the local service representative or the factory for further details.

CODE	MESSAGE	DESCRIPTION
700	"scheduler can't malloc memory" "can't create exchange" "scheduler can't create msg partition" "scheduler can't attach to DUTX" (also see ERR_send_x messages at end of this file)	error reading/writing DUTX exchange, display dead
701	"can't create exchange" (also see ERR_send_x messages at end of this file)	error reading/writing DUTC exchange, display dead
702	"CANNOT START DISPLAY SCHEDULER"	error spawning display scheduler
703	(also see ERR_send_x messages at end of this file)	error sending message to command processor - fp dead
704	"DS_get_memory: ran out of display memory."	out of display memory, display dead
705	"CANNOT INIT CPR MSG XCHANGE"	error reading/writing CPRX exchange, command processor is dead
706	"CANNOT INIT MEN CRIT MSG XCHANGE"	error reading/writing MENC exchange, menu manager dead
707	"MEMORY ALLOC ERROR FOR DS_VTEXTS"	cannot allocate memory for DS_VTEXTS in menu manager
708	"MEMORY ALLOC ERROR FOR FIELD_TEXT_VAL_P"	error allocating/freeing memory segments
709	cannot install file (INSTALL application)	
710	(also see ERR_send_x messages at end of this file)	error sending message to DFM from trace manager
711	"FATAL DFM ERROR: EXCHANGE FAILURE" "FATAL DFM ERROR: CAN'T CREATE EXCHANGE"	error reading/writing DFMX exchange, DFM dead

712	"DFM ERROR: CAN'T SEND MESG TO DSK" "DFM ERROR: CAN'T SEND MESG TO ACQ" "DFM ERROR: CAN'T SIGNAL EVENT TO TRM" "DFM ERROR: CAN'T SIGNAL COMP PROC" "DFM ERROR: CAN'T SUSPEND COMP PROCESS" "DFM ERROR: CAN'T RESUME COMP PROCESS" "DFM ERROR: CAN'T SIGNAL EVENT TO DFM" "DFM ERROR: CAN'T FIND MY OWN PID" "DFM ERROR: CAN'T SIGNAL EVENT TO HC"	error sending message/event from DFM to another process
	"DFM ERROR: CAN'T SIGNAL EVENT TO CPR" "DFM ERROR: CAN'T SEND MESG TO PIP" "DFM ERROR: WAVE DESC CHG WHEN HAVE PLUGIN" "DFM ERROR: CAN'T SEND MESSAGE TO CP"	
713	"TRE ERROR: CAN'T SEND MESG TO DFM" "TRE ERROR: CAN'T SIGNAL EVENT TO DFM" "TRE ERROR: CAN'T GET BUF FROM PARTITION"	error sending new trace eqn message to DFM
714	"COP ERROR: CAN'T FIND MY OWN PID"	error signalling event to DFM
715	"COP ERROR: CAN'T SIGNAL EVENT TO DFM"	error allocating/assigning buffer by COP
716	"DFM ERROR: CAN'T RET BUF TO PARTITION" "DFM ERROR: CAN'T ALLOC SEG FOR PARTITION" "DFM ERROR: CAN'T GET BUF FROM PARTITION" "DFM ERROR: CAN'T ALLOCATE BINNED BUFFER"	error allocating/freeing memory segment
717	not used	
718	"TIMED OUT WAITING FOR ACQUISITION SYSTEM" "TIMED OUT WAITING FOR BATTERY RAM"	timed out waiting for acqusition manager
719	"FATAL UNT ERROR: CANNOT CREATE EXCHANGE" "FATAL UNT ERROR: EXCHANGE FAILURE"	error reading/writing UNTC exchange, command procr dead, error reading/writing IMPX exchange
720	"CPR ERROR: CAN'T FIND MY PID" "OM ERROR: CAN'T FIND MY PID" "OM ERROR: CAN'T FIND DFM PID"	command processor cannot find process id, cannot function
721	"CP: CAN'T SIGNAL IM"	error signaling IM process

722	"DSK ERROR: CAN'T SEND MESSG TO DFM" "DSK ERROR: CAN'T SIGNAL EVENT TO DFM" "DSK ERROR: CPR CAN'T SEND MESSG TO DSK" "DSK ERROR: CPR CAN'T SIGNAL EVENT TO DSK" "DSK ERROR: CAN'T SIGNAL EVENT TO CPR" "DSK ERROR: CAN'T SEND MESSG TO CPR" "FATAL DSK ERROR: CAN'T FIND MY OWN PID" "FATAL DSK ERROR: CAN'T CREATE EXCHANGE" "FATAL DSK ERROR: EXCHANGE FAILURE" "DSK ERROR: CAN'T MALLOC BUFFER" "DSK ERROR: INVALID WAVE POINTER TYPE" "DSK ERROR: CAN'T SIGNAL EVENT TO IMAN"	error with interprocess communication
723	"CANNOT INIT OM MSG XCHANGE"	error reading/writing OMPX exchange
724	not used	
725	not used	
726	not used	
727	"PP ERROR: CAN'T ALLOCATE PP BUFFER" "PP ERROR: CAN'T SIGNAL EVENT TO TRM" "PP ERROR: CAN'T SIGNAL EVENT TO DFM" "PP ERROR: CAN'T SEND MESSG TO DFM" "PP ERROR: CAN'T MALLOC BUFFER" "PP ERROR: CAN'T GET BUF FROM PARTITION"	error in interprocess communication
728	"PLUGIN %c CAN'T ALLOC SEG FOR TMS CODE"	Error Allocating memory for TMS CODE
729	not used	
730	not used	
731	not used	
732	"PCON BUS TIMEOUT: ADDR 0x%x"	PCON bus timeout, restart system
733	"OM: CAN'T ALLOC SEG FOR RESPONSE BUFFER" "CP ERROR: CAN'T ALLOC SEG FOR DISK BLOCK" "OM: MEMORY ALLOC ERROR FOR BRAM STRUCT" "MEMORY ALLOCATION ERROR FOR EXT DESC" "MEMORY ALLOCATION ERROR FOR TIME ARRAY"	Error allocating/freeing/assigning memory segment
734	"CP ERR:CAN'T SEND MESSAGE TO OM" "IM ERR: CAN'T SEND MSG TO OM"	Error reading/writing other message exchanges

735	"TRE: MEMORY ALLOC ERROR FOR OPT LIST" "TRE: MEMORY ALLOC ERROR FOR PP OPT LIST" "TRE: MEMORY ALLOC ERROR FOR TRE_NODE"	Error allocating/freeing memory segment
736	"Can't send message to CP"	Error reading/writing CPRX exchange
737	"OM ERROR: CAN'T SEND MESSG TO DFM"	Error reading/writing DFMX exchange
738	"OM ERROR: CAN'T SIGNAL EVENT TO DFM"	Error signaling DFM process
739	"HC: ERROR SENDING MSG TO DFM=%d"	Error reading/writing other message exchanges
740	"HC ERROR: CAN'T FIND MY PID" "HC: CAN'T SIG DFM: NEW CMD"	Error signaling other processes
741	"CP: CAN'T ALLOC SEG FOR CMP DSPL MEM & PLOT BUF" "HC: CAN'T ALLOC SEG FOR PROGRAM BUFFER" "MEMORY ALLOC ERROR FOR LINES TO SKIP"	Error allocating/freeing memory segment
742	"PL_Cont: Fatal Error %d in Plot Preparation" "PL_Cont: Error %d"	Error during plotting or printing
743	"IM ERROR: CAN'T ALLOC INPUT BUFFER" "IM: CAN'T ALLOC SEG FOR AP INPUT BUFFER"	Error allocating/freeing/assigning memory segment
744	"ERROR SENDING MSG TO DSK=%d" "CANNOT INIT IM MSG EXCHANGE" "CAN'T SEND MESSAGE TO CP"	Error reading/writing other message exchanges
745	"IM ERROR: CAN'T FIND MY PID" "IM ERROR: CAN'T FIND CPR PID" "CAN'T SIGNAL DSK"	Error signaling events to other processes
746		Error reading/writing other message exchanges
747	"CP ERR:Can't send SER CONFIG to OM" "OM: CAN'T SIG DFM: NEW CMD" "OM: CAN'T SIG DFM: ABORT STORE CMD"	Error signaling events to other processes
748	"CP ERROR: CAN'T ALLOC SEG" "CP: CAN'T ALLOC SEG FOR INSPECT KEYWORD" "CP: CAN'T ALLOC SEG FOR PRINT BUFFER" "MEMORY ALLOC ERROR FOR TEXT STRING"	Error allocating/freeing/assigning memory segments a. trying to get memory for outputing query response
749	"CANNOT FIND A REMOTE HOST"	Controller is not GPIB or RS232 -> invalid condition
750	"ACCESS TO DEVELOPMENT VERSION DENIED"	Software development security key not present
751	PCON doesn't seem to be running	
752	"MODULE DRIVER IS INCOMPATIBLE WITH MAINFRAME");	Module driver is incompatible with mainframe code

753	error reading/writing MEM exchange	
754	"HARDCOPY CAN'T ABORT TRACE DATA OUTPUT"	During hardcopy of waveform data, the hardcopy was aborted but the HC manager could not signal the DFM to stop; therefore, you must wait for the hardcopy to complete.

MAINFRAME ERRORS

CODE	MESSAGE	DESCRIPTION
1100	"REFERENCE CLOCK NOT PRESENT"	
1101	"INCORRECT FRONT PANEL"	7200A only: incorrect front panel key scanner

PLUG-IN MESSAGES

1200	"PLUGIN %c - CHANNEL %s OVERLOAD"	input channel was overloaded
1205	"%c%d: CALIBRATION ERROR %08x"	Calibration errors. These errors get logged but do not get displayed if dongle not present. The 8 digit hexadecimal is a bit vector, with each bit indicating a different error. Values are described in pxxclerr.h.
1206	"PLUGIN %c - CAN'T HALT/UNHALT PLUGIN" "%c%d: TMS PROCESSING TIMEOUT" "%c%d: TMS PROCESSING ERROR" "%c%d: ERROR WRITING TO TMS PROGRAM RAM"	mainframe to plugin communications problem
1207	"PLUGIN %c EEPROM CONTAINS INVALID CHECKSUM"	At initialization the checksum stored in EEPROM did not match calculated one. Default settings are loaded.
1208	"PLUGIN %c NO or LOW BATTERY"	For sandia modules only, Battery Ram is checked at initialization and upon entering protected mode

Note: If Plug-in is located in slot A in the mainframe, error # will restart at 1200
If Plug-in is located in slot B in the mainframe, error # will restart at 1300

APPENDIX B: Utilities

Supplied with the LeCroy 7200 Precision Digital Oscilloscope is a Utilities Diskette containing support programs which can be executed on any IBM-compatible PC running MSDOS.

The disk includes:

COMPILE.EXE
7200TALK.EXE
WAVETRAN.EXE
GRAPH.EXE
ICLUTIL.EXE

The programs are described below.

COMPILE.EXE Compile.exe is the compiler for 7200 ICL programs. It translates ICL programs written on a PC into a form the 7200 can understand. To translate a program in the file filename.src,
type: **COMPILE FILENAME.**
The file filename.apd will be created and will contain a translated version of the program which can then be loaded into the 7200 via 3.5" floppy disk. See Section 7 of the 7200 Remote Programmer's Manual.

7200TALK.EXE 7200talk.exe is a GPIB communication program. It is self-prompting and contains built-in help.

7200talk is compatible with the National Instruments PCII, PCIIA, and PCIII GPIB cards and uses the National Instruments device driver gpib.com.

WAVETRAN.EXE Wavetran.exe translates a binary waveform file created by a LeCroy 7200 Precision Oscilloscope and generates ASCII (readable text) output.
Type: **WAVETRAN -h** for a help message.
To print the data values in the waveform file trace1.000,
type: **WAVETRAN TRACE1.000.**
The data values will be printed to the screen. To direct the output to the file data,
type: **WAVETRAN -oDATA TRACE1.000.**
To print the acquisition control settings and other information associated with the waveform,
type: **WAVETRAN -d TRACE1.000**.

The acquisition information will be printed to the screen. To direct the output to the file info,
type: **WAVETRAN -d -oINFO TRACE1.000.**

Wavetran uses another file, called the template file, to interpret the waveform. The template file describes the structure of the waveform produced by the oscilloscope. The file 7200.tpl on the Utilities Diskette contains the template for waveforms produced by the 7200 oscilloscope. (The template may be read out of the 7200 via GPIB or RS-232C with the TEMPLATE? remote query, but is provided on the Utilities Diskette in the file 7200.tpl for convenience). Wavetran uses the template file named 7200.tpl by default.

To translate waveforms produced by another LeCroy oscilloscope, a different template file may be specified by adding the -tTEMPLATE option to the command line.

> *IMPORTANT: If wavetran is copied onto another disk, it will not work unless the file 7200.tpl is also copied into the same directory.*

See Section 3 of the 7200 Remote Programmer's Manual for a description of the 7200 waveform format and the template.

The information and data can be printed in different formats by specifying a print format file, as **WAVETRAN -fFORMAT.FMT TRACE1.000.** The file all.fmt on the Utilities Diskette is a sample format file that prints all parts of a waveform, including sequence trigger times.

Each line of the format file is either a command or the name of a variable (containing acqusition information - eg. VERTICAL_GAIN) to be printed. Lines starting with '$' are commands. The commands are:

$PRINT WAVEFORM
prints the entire waveform
$PRINT BLOCK <block name> DEFAULT
prints all variables in the specified block in default order
$PRINT BLOCK <block name> SPECIFIED
prints the variables in the block which appear on subsequent lines of the format file
$PRINT ARRAY <array name><start point><end point>
prints the specified data or time array from start point to end point
end point = -1 means print to the end
$COLS <num columns>

specifies number of columns for output format

$LINES <number lines>

specifies number of lines to skip between each line of output

$LABELS <ON or OFF>

specifies whether or not to label the parts of the waveform

$SKIP <num lines>

skip the specified number of lines at the point in this output

$TITLE <title string>

print the specified title string at the point in this output

GRAPH.EXE

Graph.exe graphs a list of up to 500 numbers on the screen of a PC. Type: **GRAPH -h** for a help message.

To graph the data points in a file called data (which may have been created by using wavetran to translate a 7200 waveform), type: **GRAPH DATA**.

ICLUTIL.EXE

ICLUTIL.EXE is a tool which allows you to create, edit, view, or print ICL programs. In addition, it allows you to transfer ICL programs between a personal computer and the LeCroy 7200 Precision Digital Oscilloscope.

The ICL Utility is designed to run on an IBM Personal Computer or compatible, which is running DOS 3.0 or greater, and has any of the following communication port combinations:

a. 1 or more RS-232 serial ports and 1 or more Centronics compatible parallel printer ports.

b. 1 or more RS-232 serial ports and 1 or more National Instrumentsmodel PCII, PCIIA, or PCIII GPIB cards.

c. 2 or more RS-232 serial ports.

d. 1 or more National Instruments model PCII, PCIIA, or PCIII GPIB cards.

The following software drivers must be loaded into the computer during the boot process:

a. ANSI.SYS or compatible

b. If a National Instruments GPIB card is installed, the device driver GPIB.COM should be installed

Installation of the device drivers requires that the files named above be located in the root directory of the disk used to boot the com-

puter. The device drivers will be loaded into memory provided that a CONFIG.SYS file which includes these device driver filenames is also present in the root directory.

An example of a CONFIG.SYS and GPIB.COM is shown below:

```
FILES=10
BUFFERS=10
DEVICE=ANSI.SYS
DEVICE=GPIB.COM
```

The ICLUTIL.EXE program should be present in the same directory as the ICLUTIL.EXE programs which it creates, edits, and transfers.

In addition, the ICL compiler program, COMPILE.EXE, should be present in the same directory.

If these executable files are not located in the same directory, then both a PATH and APPEND statement should be added to an AUTOEXEC.BAT file in the root directory of the disk used to boot the computer. These statements will inform the computer where to find these programs. The following is a typical example:

Suppose you wish to separate the utility programs from your ICL programs. The utility programs might be stored in the directory c:\7200UTIL and the ICL programs might be stored in the directory C:\ICLPROG. To invoke the ICL Compiler or the ICL Programming Utility from the directory c:\ICLPROG, the following two lines must be present in an AUTOEXEC,BAT file in the root directory c:\

```
PATH=C:\7200UTIL
APPEND C:\7200UTIL
```

The ICL Utility may be executed by simply typing ICLUTIL, which causes the Main Menu to be displayed. The boxes displayed on the left side if the Main Menu correspond to the function keys <F1> through <F10> on your computer to perform the task associated with it.

For example, pressing the function key <F1> will recall some helpful information about the program's features.

Prior to attempting to communicate with a LeCroy 7200 or a printer, the program should be configured by pressing <F2> while the Main

Menu is displayed. The communication setup which is displayed, should agree with the remote control configuration of the LeCroy 7200. The computer communications setup may be modified by pressing the arrow keys on the keyboard. Press the <Esc> key when all changes are completed. The program will remember to use the new setup whenever it is executed.

To transfer programs between your computer and the 7200, you must set the 7200's Remote Communication port to the port (GPIB or RS232) you wish to use. Refer to section 1 of this manuals for more details and cabling instructions.

7200A Index

D

E

K

L

M

N

O

P

Q

R

S

T

U

V

W

X