Remote Control Manual

LeCroy
LSA1000 *Signalyst*

Revision C — March 2000

LeCroy

Corporate Headquarters
700 Chestnut Ridge Road
Chestnut Ridge, NY 10977–6499
Tel: (914) 578 6020, Fax: (914) 578 5985

European Manufacturing
2, rue du Pré-de-la-Fontaine
1217 Meyrin 1/Geneva, Switzerland
Tel: (41) 22 719 21 11, Fax: (41) 22 782 39 15

Internet: www.lecroy.com

LSA1000-RCM Rev C 0300

Contents

Explaining the LSA1000

There are two manuals that explain the LSA1000. The accompanying *Operator's Manual* takes you through the initial steps and gets you started using the instrument. It explains basics such as how to connect to a PC and use the software tools supplied. Once familiar with the LSA1000's basic operation, use this, the *Remote Control Manual.* It contains detailed descriptions of all the remote commands used to operate the LSA1000 from the computer.

About This Manual…

- ***Chapter 1*** describes warranty, maintenance agreements, service and return procedure.
- ***Chapter 2*** explains the overall structure of commands to control the LSA1000 from your PC.
- ***Chapter 3*** gives an overview of operation over the Ethernet, including protocol, data transfer header and problem-solving tips.
- ***Chapter 4*** Waveform Structure, covers important commands and basic rules for reading and writing waveform data with LSA1000.
- ***Chapter 5*** describes each of the registers that can be used to poll the LSA1000's internal processing status.
- ***Appendix A*** is the Waveform Template.

LSA1000: Legacy of the Oscilloscope

In this publication and its companion, the LSA1000 *Operator's Manual,* references are to be found to functions not directly applicable to the LSA1000. Some examples are references to "time/div", "cursors", and "display".

Their presence in dedicated LSA1000 manuals is owing to the legacy of LeCroy DSOs (Digital Storage Oscilloscopes) in the development of the LSA1000. Although the current practicability of these functions may not immediately be apparent, the basic concepts to which they adhere remain valid for the LSA1000, and the functions are supported by remote control commands.

Moreover, in order to maintain compatibility, the LSA1000's remote commands have been made a subset of the commands for the LeCroy digital oscilloscopes.

Terminology borrowed from the oscilloscope should thus be understood to refer to the LSA1000 *conceptually*, as if it possessed an oscilloscope display. It should be noted, for example, that all commands that refer to "divisions" on a DSO are applicable to the LSA1000: there are eight divisions full scale in the vertical (voltage) direction, and 10 divisions in the horizontal (time) axis.

Foreword

Warranty and Product Support

It is recommended that you thoroughly inspect the contents of the LSA1000 packaging immediately upon receipt. Check all contents against the packing list/invoice copy shipped with the instrument. Unless LeCroy is notified promptly of any missing or damaged item, responsibility for its replacement cannot be accepted. Contact your nearest LeCroy Customer Service Center or national distributor immediately.

Warranty

LeCroy warrants this product for normal use and operation within specifications for a period of three years from the date of shipment. Calibration each year is recommended to ensure in-spec. performance. Spares, replacement parts and repairs are warranted for 90 days. The instrument's firmware has been thoroughly tested and is thought to be functional, but is supplied without warranty of any kind covering detailed performance. Products not made by LeCroy are covered solely by the warranty of the original equipment manufacturer.

Under the LeCroy warranty, LeCroy will repair or, at its option, replace any product returned within the warranty period to a LeCroy authorized service center. However, this will be done only if the product is determined after examination by LeCroy to be defective due to workmanship or materials, and not to have been caused by misuse, neglect or accident, or by abnormal conditions or operation.

***Note:** This warranty replaces all other warranties, expressed or implied, including but not limited to any implied warranty of merchantability, fitness, or adequacy for any particular purpose or use. LeCroy shall not be liable for any special, incidental, or consequential damages, whether in contract or otherwise. The client will be responsible for the transportation and insurance charges for the return of products to the service facility. LeCroy will return all products under warranty with transport prepaid.*

Maintenance Agreements

LeCroy provides a variety of customer support services under Maintenance Agreements. Such agreements give extended warranty and allow clients to budget maintenance costs after the initial three-year warranty has expired. Other services such as installation, training, enhancements and on-site repairs are available through special supplemental support agreements.

Staying Up to Date

LeCroy is dedicated to offering state-of-the-art instruments, by continually refining and improving the performance of LeCroy products. Because of the speed with which physical modifications may be implemented, this manual and related documentation may not agree in every detail with the products they describe. For example, there might be small discrepancies in the values of components affecting pulse shape, timing or offset, and — infrequently — minor logic changes. However, be assured the LSA1000 itself is in full order and incorporates the most up-to-date circuitry.

LeCroy frequently updates firmware and software during servicing to improve LSA1000 performance, free of charge during warranty. You will be kept informed of such changes, through new or revised manuals and other publications.

Nevertheless, you should retain this, the original manual, for future reference to your LSA1000's unchanged hardware specifications.

Service and Repair

Please return products requiring maintenance to the Customer Service Department in your country or to an authorized service facility. The customer is responsible for transportation charges to the factory, whereas all in-warranty products will be returned to you with transportation prepaid. Outside the warranty period, you will need to provide us with a purchase order number before we can repair your LeCroy product. You will be billed for parts and labor related to the repair work, and for shipping.

How to Return a Product

Contact the nearest LeCroy Service Center or office to find out where to return the product. All returned products should be identified by model and serial number. You should describe the defect or failure, and provide your name and contact number. In the case of a product returned to the factory, a Return Authorization Number (RAN) should be used. The RAN can be obtained by contacting the Customer Service Department.

Return shipments should be made prepaid. We cannot accept COD (Cash On Delivery) or Collect Return shipments. We recommend air-freighting.

It is important that the RAN be clearly shown on the outside of the shipping package for prompt redirection to the appropriate LeCroy department.

What Comes with LSA1000

The following items are shipped together with the standard configuration of the LSA1000:

- Getting Started Software CD-ROM
- AC Power Cord and Plug
- *Operator's Manual*
- *Remote Control Manual* (this manual)
- NIST Calibration Certificate
- Declaration of Conformity
- Warranty.

Technical Assistance…

Help on installation, calibration, and the use of LeCroy equipment is available from the LeCroy Customer Service Center in your country (*see contact numbers following index*). *See also below.*

Hardware Assistance…

Before contacting us on hardware-related questions (*see below*), try the following:

- ➢ Review Chapter 2 of the *Operator's Manual*, which covers general technical concepts and functions such as acquisition, triggering, and memory.
- ➢ See Appendix A of the *Operator's Manual* for the LSA1000's technical specifications.

Software Assistance…

Before contacting us with software-related questions, try the following:

- ➢ Review Chapter 5 of the *Operator's Manual*, which contains several program examples that can be used as a starting point when writing application-specific software using the LSA1000.
- ➢ See the *Remote Control Manual* for questions on specific commands, or if unsure as to which command to use.
- ➢ Check *Frequently Asked Questions* at the LeCroy web site: www.lecroy.com

… Or Contact

LeCroy Embedded Signal Analysis:

E-mail: esa@lecroy.com

Fax: ++1 914 578 4485

Phone: (800) 5-LeCroy

1 Read This First!

Controlling the LSA1000

The LeCroy LSA1000 is operated by remote control using a controlling device, normally a computer but perhaps a simple terminal. Connected via Ethernet, it has a TCP/IP port, but also possesses a USB (Universal Serial Bus) port, to be supported by software in the future.

The only actions of the LSA1000 that cannot be performed remotely are power on or off.

This chapter introduces the basic concepts affecting the instrument's operation, while the following chapter explains how it operates through the Ethernet. Chapter 4 offers a detailed description and run-through of the transfer and formatting of waveforms. And Chapter 5 explains the use of status bytes for error reporting.

The special *System Commands* section provides a complete directory and description of the remote control commands and queries that can be used to operate the LSA1000.

Implementation Standard

To the greatest extent possible, these remote commands conform to the IEEE 488.2* standard, which may be considered as an extension of the IEEE 488.1 standard, dealing mainly with electrical and mechanical issues.

Program Messages

Program messages sent to the LSA1000 from the external controller must conform to precise format structures. The instrument will execute such messages, but will ignore program messages in which errors are detected.

Warning or error messages are normally not reported unless the controller explicitly examines the relevant status register. Or if the status-enable registers have been set so that the controller can be interrupted when an error occurs.

* ANSI/IEEE Std. 488.2–1987, *IEEE Standard Codes, Formats, Protocols, and Common Commands*. The Institute of Electrical and Electronics Engineers Inc., 345 East 47th Street, New York, NY 10017, USA.

Commands and Queries

Program messages consist of either one or several commands or queries or both. A command directs the instrument to change its state — for example, to change its timebase or vertical sensitivity. A query asks the instrument about its state. Very often, the same mnemonic is used for a command and a query, the query being identified by a <?> after the last character.

For example, to change the timebase to 2 ms/div, the controller sends the following command to the instrument:

```
TIME_DIV 2 MS
```

To ask the instrument about its timebase, this query should be sent:

```
TIME_DIV?
```

A query causes the instrument to send a response message. The control program should read this message with a 'read' instruction to the ETHERNET interface of the controller. The response message to the query above might be:

```
TIME_DIV 10 NS
```

The portion of the query preceding the question mark is repeated as part of the response message. If desired, this text may be suppressed with the command "COMM_HEADER".

Depending on the state of the instrument and the computation to be done, the controller may have to wait up to several seconds for a response. Command interpretation does not have priority over other LSA1000 activities. It is therefore judicious to set the controller IO timeout conditions to three or more seconds. In addition, it must be remembered that an incorrect query message will not generate a response message.

Program Message Form

An instrument is remotely controlled with program messages that consist of one or several commands or queries, separated by semicolons <;> and ended by a terminator:

<command/query>;.........;<command/query> <terminator>

Upper or lower-case characters or both can be used in program messages.

The instrument does not decode an incoming program message before a terminator has been received, except if the program message is longer than the 256 byte input buffer of the instrument, when the LSA1000 starts analyzing the message when the buffer is full. The commands or queries are executed in the order in which they are transmitted.

See Chapter 3 for a description of LeCroy's Versatile Instrument Control Protocol (VICP) for operation over ETHERNET.

Example

`ARM`

This program message consists of a single command that instructs the instrument to change its state from "stopped" to "single". The terminator is not shown, as it is assumed to be automatically added by the interface driver routine.

```
        COMB 2; ARM; DATE?
```

This program message consists of two commands, followed by a query. They instruct the instrument to combine channels, arm the acquisition, and then ask for the current date. Again, the terminator is not shown.

Command/Query Form

The general form of a command or a query consists of a command header <header> optionally followed by one or several parameters <data> separated by commas:

<header>[?] <data>,...,<data>

The notation [?] shows that the question mark is optional (turning the command into a query). The detailed listing of all commands in *System Commands* indicates which may also be queries. There is a space between the header and the first parameter. There are commas between parameters.

Example

`DATE 15,JAN,1998,13,21,16`

This command instructs the LSA1000 to set its date and time to 15 JAN 1998, 13:21:16. The command header "DATE" indicates the action, the 6 data values specify it in detail.

Header

The header is the mnemonic form of the operation to be performed by the LSA1000. All command mnemonics are listed in alphabetic order in the System Commands section.

The majority of the command or query headers have a long form for optimum legibility and a short form for better transfer and decoding speed. The two forms are fully equivalent and can be used interchangeably. For example, the following two commands for switching to the normal trigger mode are fully equivalent:

`TRIG_MODE NORM` and **`TRMD NORM`**

Some command/query mnemonics are imposed by the IEEE 488.2 standard. They are standardized so that different instruments present the same programming interface for similar functions. All these mnemonics begin with an asterisk <*>. For example, the command "*RST" is the IEEE 488.2 imposed mnemonic for resetting the instrument, whereas "*TST?" instructs the instrument to perform an internal self-test and to report the outcome.

Header path

Some commands or queries apply to a sub-section of the LSA1000 — a single input channel or a trace, for example. In such cases, the header must be preceded by a path name that indicates the channel or trace to which the command applies. The header path normally consists of a two-letter path name followed by a colon <:> immediately preceding the command header.

One of the waveform traces can usually be specified in the header path (refer to the individual commands listed in *System Commands* for details of the values applying to given command headers):

C1, C2	Channels 1 and 2
M1, M2, M3, M4	Memories 1, 2, 3, 4
TA, TB, TC, TD	Traces A, B, C and D
EX	External trigger

Example

`C1:OFST -300 MV`

Set the offset of Channel 1 to –300 mV

Header paths need only be specified once. Subsequent commands whose header destination is not indicated are assumed to refer to the last defined path. For example, the following commands are identical:

`C2:VDIV?; C2:OFST?`	What is the vertical sensitivity and the offset of channel 2?
`C2:VDIV?; OFST?`	Same as above, without repeating the path.

Data

Whenever a command/query uses additional data values, the values are expressed in terms of ASCII characters. There is a single exception: the transfer of waveforms with the command/query "WAVEFORM", where the waveform may be expressed as a sequence of binary data values. Chapter 5 gives a detailed explanation of the format of waveforms.

ASCII data can have the form of character, numeric, string or block data.

Numeric Data

The numeric data type is used to enter quantitative information. Numbers can be entered as integers or fractions, or in exponential representation.

`C2:OFST 3.56`	Set the DC offset of Channel 2 to 3.56 V.
`TDIV 5.0E-6`	Adjust the timebase to 5 µs/div.

***Note:** Numeric values may be followed by multipliers and units, modifying the value of the numerical expression. The following mnemonics are recognized:*

EX	1E18	Exa-	PE	1E15	Peta-
T	1E12	Tera-	G	1E9	Giga-
MA	1E6	Mega-	K	1E3	kilo-
M	1E-3	milli-	U	1E-6	micro-
N	1E-9	nano-	PI	1E-12	pico-
F	1E-15	femto-	A	1E-18	atto-

Examples

There are many ways of setting the timebase of the instrument to 5 µs/div:

`TDIV 5E-6`	Exponential notation, without any suffix.
`TDIV 5 US`	Suffix multiplier "U" for 1E–6, with the (optional) suffix "S" for seconds.

or

```
TDIV 5000 NS
TDIV 5000E-3 US
```

Block Data

These are binary data values coded in hexadecimal ASCII, i.e. 4-bit nibbles are translated into the digits 0,...9, A,...F and transmitted as ASCII characters. They are used only for the transfer of waveforms (command "WAVEFORM") and of the instrument configuration

Response Message Form

The instrument sends a response message to the controller, as an answer to a query. The format of such messages is the same as that of program messages, i.e. individual responses in the format of commands, separated by semicolons <;> and ended by a terminator. They can be sent back to the instrument in the form in which they are received, and will be accepted as valid commands.

For example, if the controller sends the program message:

`TIME_DIV?;TRIG_MODE NORM;C1:VDIV?`

(terminator not shown).

The instrument might respond as follows:

`TIME_DIV 50 NS;C1:VDIV 125mV` (terminator not shown).

The response message refers only to the queries: "TRIG_MODE" is left out. If this response is sent back to the instrument, it is a valid program message for setting its timebase to 50 ns/div and the input coupling of Channel 1 to 50 Ω.

Whenever a response is expected from the instrument, the control program must instruct the ETHERNET interface to read from the instrument.

The instrument uses somewhat stricter rules for response messages than for the acceptance of program messages. Whereas the controller may send program messages in upper or lower case characters, response messages are always returned in upper case. Program messages may contain extraneous spaces or tabs (white space); response messages do not. And while program messages may contain a mixture of short and long command/query headers, response messages always use short headers as a default. However, the instrument can be forced with the command "COMM_HEADER" to use long headers or no headers at all. If the response header is omitted, the response transfer time is minimized, but such a response could not be sent back to the instrument again. In this case suffix units are also suppressed in the response.

If the trigger slope of Channel 1 is set to negative, the query “C1:TRSL?” could yield the following responses:

`C1:TRIG_SLOPE NEG`	header format: long
`C1:TRSL NEG`	header format: short
`NEG`	header format: off

Waveforms which are obtained from the instrument using the query “WAVEFORM?” constitute a special kind of response message. Their exact format can be controlled via the “COMM_FORMAT” and “COMM_ORDER” commands.

2 Overview of LSA1000 Control

Operating on the Ethernet

All LSA1000 functions are controlled via Ethernet, the LAN (Local Area Network) software standard. The instrument uses Ethernet's TCP/IP network protocol, accessed by the BSD Sockets API. *For connecting to PC or network over the Ethernet, see also Chapter 4 of the accompanying Operator's Manual.*

This API (Applications Programming Interface) sits above the TCP/IP protocol stack in all UNIX systems and is also available in Windows 95 and Windows NT. It is for the most part platform-independent, and should allow the same source code to compile and run on each of the supported systems.

WinSock

The commonly known *WinSock* API, derived from the original BSD Sockets API, may also be used to communicate with BSD Sockets-based systems. *WinSock*, for *Windows Sockets,* is used by the leading Internet servers.

TCP (Stream Socket)

Of the two types of connections supported by BSD Sockets, UDP and TCP, the LSA1000 uses TCP — also called *stream socket* — as its underlying protocol. This is a 'reliable' protocol, which ensures that packets are in the correct sequence and that none are missing.

VICP

The Versatile Instrument Control Protocol (VICP) is the LSA1000 protocol for Ethernet operation. The connection established between the controlling device, or *client*, and the LSA1000, or *server*, is made using a known port number. Each of the common Internet protocols uses a predefined port number — FTP, for example, uses 21, and HTTP 80. **The VICP port number is 1861.**

Note: A USB (Universal Serial Bus) port is located on the instrument's rear panel even though USB communication is not supported at this time. It is intended that this communication protocol will be supported in future LeCroy software releases.

The client sends standard ASCII remote commands through the Ethernet socket, just as they would be sent via GPIB, but with an 8-byte header at the start of each transfer. This header contains information about the type of block and its length. Block types include 'Data with/without EOI', and Device Clear, and allow GPIB behavior to be emulated.

Addressing

Every Ethernet device has an IP address designated by four numbers between 0 and 255, separated by periods — for example, 12.34.56.78. Your LSA1000's address is set to 172.25.1.2 at the factory but can be changed using the COMM_NET command.

Standard Messages

The following are IEEE 488.1 standard messages that go beyond mere reconfiguration of the bus and that have an effect on the operation of the instrument. All except GET are executed immediately upon reception — not in chronological order.

- In response to a universal **Device CLear** (DCL) or a Selected Device Clear message (SDC), the LSA1000 clears the input or output buffers, aborts the interpretation of the current command (if any) and clears any pending commands. Status registers and status-enable registers are not cleared. Although DCL has an immediate effect it can take several seconds to execute this command if the instrument is busy.
- The **Group Execute Trigger** message (GET) causes the LSA1000 to arm the trigger system. It is functionally identical to the "*TRG" command.

Programming Ethernet Transfers

Data Transfer Header

The format of the header sent before each data block, both to and from the LSA1000, is set out in the following table:

Byte #	Purpose
0	Operation
1	Header Version
2	Spare (reserved for future expansion)
3	Spare (reserved for future expansion)
4	Block Length, (bytes of data), MSB
5	Block Length (bytes of data)
6	Block Length (bytes of data)
7	Block Length, (bytes of data), LSB

The ‘Operation’ bits and meanings are:

D7	D6	D5	D4	D3	D2	D1	D0
DATA	REMOTE	LOCKOUT	CLEAR	SRQ	Reserved	Reserved	EOI

Data Bit	Mnemonic	Purpose
D7	DATA	Data block (D0 indicates termination with/without EOI)
D6	REMOTE	Remote Mode
D5	LOCKOUT	Local Lockout (Lockout front-panel)
D4	CLEAR	Device Clear (if sent with data, clear occurs before data block is passed to parser)
D3	SRQ	SRQ (Device to PC only)
D2..D1	Reserved	Reserved for future expansion
D0	EOI	Block terminated in EOI Logic "1" = use $\rightarrow$ EOI terminator Logic "0" = no EOI terminator

It is possible that the LSA1000 and the controlling application will get out of sync with each other. For this, a recovery mechanism has been defined, and the controller at the end of the connection that detects the problem is responsible for closing the socket and re-opening it.

Problem Solving

The TCP 'NAGLE' algorithm: One of the algorithms used in the TCP layer of the TCP/IP stack is the cause of important remote control performance problems. This algorithm has the function of buffering up small packets and sending them only when a 'large' packet has been filled or a time limit of 200 ms has expired. Even a simple query is bound by these limitations.

However, when NAGLE is turned off, this 'round-trip' time is reduced by approximately one hundred. The following function call disables the algorithm when using the standard BSD Sockets API of the 'C' language (equivalent function calls may exist in other environments).

```
const int disable = 1;

if    (0    !=    setsockopt(socket,    IPPROTO_TCP,
TCP_NODELAY, (char*)&disable, sizeof(disable)))

{

    … failed …

}
```

Multiple Client Support: The current design of network remote control allows support of only one client at a time. This applies equally to the operation of the LSA1000*.* And because of this the number of simultaneous connections that can be made with the instrument has been restricted to one.

This can cause problems if a remote client disconnects or hangs without closing its connection (socket). Unfortunately there is no 'clean' way for the server to know when this has happened. If the LSA1000 seems to be refusing connections then a reboot may be required.

This problem is due to be addressed in a future revision of the protocol.

Problem Solving

C' Language: The following sample 'C' code allows a simple dialog to be established with the LSA1000. The sockets are used in a blocking mode (processing is suspended while a response is awaited). Non-blocking operation is beyond the scope of this manual, but is covered in almost any BSD sockets reference.

```
/*------------------------------------------------------------
------------------

	LeCroy LSA1000 BSD Sockets Remote Control Example


	Overview:
		This example shows how to send a remote query
to a LSA1000
		and read it's response. It should be used as
a model for more
		complex remote control systems.

	Requirements:
		Microsoft Visual C++ 4.x, 5.0 compiler
		Windows 95/NT host

	Version: 1.0, August 14th

	Notes:
		Ensure  that  the  SERVER_ADDRESS  correctly
reflects the address of
		the device under control.

  ----------------------------------------------------------
------------------*/

#include "windows.h"
#include <stdio.h>

#define SERVER_PORT		1861
#define SERVER_ADDRESS		"172.25.1.2"
#define HEADER_LENGTH		8

#define FLAG_EOI		0x80 + 0x01
#define FLAG_NO_EOI		0x80

int socketFd;			/* client socket handle */

/* function prototypes */
BOOL connectToScope();
void disconnectFromScope();
int readString(char *replyBuf, int userBufferSize);
BOOL  sendString(char  *message,  int  bytesToSend,  BOOL
eoiTermination);

/* main: program entry point */
int main()
```

```
{
    char replyBuf[81];

    connectToScope();
    sendString("*idn?\n", 6, TRUE);
    readString(replyBuf, 80);
    disconnectFromScope();

    printf("Scope's reply: [%s]\n", replyBuf);

    return(0);
}

/* connectToScope: connect to a network device */
BOOL connectToScope()
{
    SOCKADDR_IN       serverAddr;              /* server's
socket address */
    int sockAddrSize = sizeof (SOCKADDR);        /* size of
socket address structures */

    /* one-time initialization of WinSock
       (not required on UNIX platforms) */
       int err;
       WORD wVersionRequested = MAKEWORD(1, 1);
       WSADATA wsaData;

       err = WSAStartup(wVersionRequested, &wsaData);
       if (err != 0)
       {
           printf("ERROR: could not initialize WinSock\n");
           return(FALSE);
       }

    /* build server socket address */
       serverAddr.sin_family = AF_INET;
       serverAddr.sin_port = htons (SERVER_PORT);

       if          ((serverAddr.sin_addr.s_addr          =
inet_addr(SERVER_ADDRESS)) == -1)
       {
           printf("ERROR: Bad server address\n");
           return(FALSE);
       }

    /* create client's socket */
       socketFd = socket(AF_INET, SOCK_STREAM, 0);
       if (socketFd == INVALID_SOCKET)
       {
             printf("ERROR:  socket()  failed,  error  code  =
%d\n", WSAGetLastError());
           return(FALSE);
       }

    /* connect to server (scope) */
```

```
        if ((connect(socketFd, (SOCKADDR FAR *) &serverAddr,
sockAddrSize)) == SOCKET_ERROR)
        {
              printf("ERROR:  socket()  failed,  error  code  =
%d\n", WSAGetLastError());
            return(FALSE);
        }

    /* success */
        return(TRUE);
}

/* disconnectFromScope: disconnect from a network device */
void disconnectFromScope()
{
    closesocket(socketFd);
}

/* sendString: send a string to the device, with or without
EOI termination */
BOOL   sendString(char   *message,   int   bytesToSend,   BOOL
eoiTermination)
{
    static unsigned char headerBuf[HEADER_LENGTH];
    int bytesSent;

    /* send header */
        if(eoiTermination)
            headerBuf[0] = FLAG_EOI;
        else
            headerBuf[0] = FLAG_NO_EOI;
        headerBuf[1] = 1;                    /*       header
version 1 */
        headerBuf[2] = 0x00;          /* unused */
        headerBuf[3] = 0x00;          /* unused */
        *((unsigned     long     *)     &headerBuf[4])     =
htonl(bytesToSend);   /* message size */

        if     (send(socketFd,     (char     *)     headerBuf,
HEADER_LENGTH, 0) != HEADER_LENGTH)
        {
            printf("ERROR: could not send header\n");
            return(FALSE);
        }

    /* send contents of message */
        bytesSent = send(socketFd, message, bytesToSend, 0);
        if   ((bytesSent   ==   ERROR)   ||   (bytesSent   !=
bytesToSend))
        {
            printf("ERROR: 'send' failed\n");
            return(FALSE);
        }

   return(TRUE);
}
```

```
/* readString: read a string from the device into a user-
supplied buffer */
int readString(char *replyBuf, int userBufferSize)
{
        int blockSize = 0, thisBlockSize, bytesReceived;
        BOOL blockEOITerminated = FALSE;
        unsigned char headerBuf[HEADER_LENGTH];

        /* read the header */
                if(recv(socketFd,    (char    *)    headerBuf,
HEADER_LENGTH, 0) == 8)
                {
                     /* extract the number of bytes contained
in this packet */
                         blockSize = ntohl(*((unsigned long *)
&headerBuf[4]));

                    /* check the integrity of the header */
                         if(!((headerBuf[0]  ==  FLAG_EOI  ||
headerBuf[0] == FLAG_NO_EOI) &&
                                headerBuf[1] == 0x01))
                        {
                             /* error state, cannot recognise
                header since we
                                    are  out  of  sync,  need  to
close & reopen the socket */
                                disconnectFromScope();
                                connectToScope();
                                return(0);
                        }

                    /* inform the caller of the EOI state */
                        if(headerBuf[0] == 0xaa)
                            blockEOITerminated = TRUE;
                }

        /* read the data block */
                thisBlockSize      =      min(userBufferSize,
blockSize);

                bytesReceived   =   recv(socketFd,   replyBuf,
thisBlockSize, 0);

                if(bytesReceived != thisBlockSize)
                    printf("ERROR: truncated read\n");
                else
                    replyBuf[bytesReceived] = '\0';   /*
ensure string termination */

        return(bytesReceived);
}
```

3 The LSA1000 and the Ethernet

Understanding Waveforms

This chapter covers the reading and writing of waveforms in remote control, and attempts to explain their structure and content.

Basic Structure

Waveforms can be divided into two basic entities. One is the basic data array: the raw data values from the LSA1000's ADCs (Analog–to–Digital Converters) in the acquisition. The other is the accompanying descriptive information, such as vertical and horizontal scale, and time of day, necessary for a full understanding of the information contained in the waveform.

This information can be accessed by remote control using the INSPECT? Query, which interprets it in an easily understood ASCII text form. It can be more rapidly transferred using the WAVEFORM? query, or written back into the instrument with the WAVEFORM command.

The LSA1000 itself contains a data structure, or *template*, which provides a detailed description of how the waveform's information is organized.

Waveforms can also be stored in pre-formatted ASCII output, for popular spreadsheets and math processing packages, using the STORE and STORE_SETUP commands.

Waveform Template

This gives a detailed description of the form and content of the logical data blocks of a waveform, and is provided as a reference. Although a sample template is given elsewhere in this manual (*see Appendix A*), it is suggested that the TEMPLATE? query and the actual instrument template be used. The template may change as the instrument's firmware is enhanced, and it will help provide backward compatibility for the interpretation of waveforms.

Logical Data Blocks

A waveform normally contains a waveform descriptor block and a data array block. However, in more complicated cases, one or more other blocks will be present.

- Waveform Descriptor block (WAVEDESC): This block includes all the information necessary to reconstitute the display of the waveform from the data. This includes:
  - hardware settings at the time of acquisition
  - the exact time of the event
  - the kinds of processing that have been performed
  - the name and serial number of the instrument
  - the encoding format used for the data blocks
  - miscellaneous constants.
- Optional User-provided Text block (USERTEXT): The WFTX command can be used to put a title or description of a waveform into this block. The WFTX? query command gives an alternative way to read it. This text block can hold up to 160 characters. They can be displayed in the TEXT + TIMES status menu as four lines of 40 characters.
- First data array block (SIMPLE or DATA_ARRAY_1): This is the basic integer data of the waveform. It can be raw or corrected ADC data or the integer result of waveform processing.
- Second data array block (DATA_ARRAY_2): This second data array is needed to hold the results of processing functions such as the Extrema (WP01 option) or Complex FFT (WP02 option). In such cases, the data arrays contain:

	Extrema	**FFT**
DATA_ARRAY_1	Roof trace	Real part
DATA_ARRAY_2	Floor trace	Imaginary part

***Note:** The Template also describes an array named DUAL. This is simply a way to allow the INSPECT? command to examine the two data arrays together.*

Using the INSPECT? Query

The query INSPECT? is a simple way to examine the contents of a waveform in remote control.

Usable on both the data and descriptive parts, its most basic form is:

INSPECT? "name**"**

where the template gives the name of a descriptor item or data block. The answer is returned as a single string, but may span many lines. Some typical dialogue:

question **C1:INSPECT? "VERTICAL_OFFSET"**
response **C1:INSP "VERTICAL_OFFSET: 1.5625e-03"**
question **C1:INSPECT? "TRIGGER_TIME"**
response **C1:INSP "TRIGGER_TIME: Date = FEB 17, 1994, Time = 4: 4:29.5580"**

INSPECT? can also be used to provide a readable translation of the full waveform descriptor block with:

INSPECT? "WAVEDESC"

The template dump will give details of the interpretation of each of the parameters. INSPECT? is also used to examine the measured data values of a waveform using:

INSPECT? "SIMPLE"

For example, for an acquisition with 52 points:

```
INSPECT? "SIMPLE"
C1:INSP "
 0.0005225   0.0006475  -0.00029    -0.000915    2.25001E-0  0.000835
 0.0001475  -0.0013525  -0.00204    -4E-05       0.0011475   0.0011475
-0.000915   -0.00179    -0.0002275   0.0011475   0.001085   -0.00079
-0.00179    -0.0002275   0.00071     0.00096    -0.0003525  -0.00104
 0.0002725   0.0007725   0.00071    -0.0003525  -0.00129    -0.0002275
 0.0005225   0.00046    -0.00104    -0.00154     0.0005225   0.0012725
 0.001335   -0.0009775  -0.001915   -0.000165    0.0012725   0.00096
-0.000665   -0.001665   -0.0001025   0.0010225   0.00096    -0.0003525
-0.000915    8.50001E-0  0.000835    0.0005225
 "
```

These numbers are the fully converted measurements in volts. Of course, when the data block contains thousands of items the string will contain a great many lines.

Depending on the application, the data may be preferred in its raw form as either a BYTE (8 bits) or a WORD (16 bits) for each data value. In this case the relations given below must be used in association with WAVEFORM? to interpret the measurement. It might then say:

```
INSPECT? “SIMPLE”,BYTE
```

The examination of data values for waveforms with two data arrays can be performed as follows:

INSPECT? “DUAL”	to get pairs of data values on a single line
INSPECT? “DATA_ARRAY_1”	to get the values of the first data array
INSPECT? “DATA_ARRAY_2”	to get the values of the second data array.

Finally…

INSPECT? is useful, but it is also a rather verbose way to send information. As a query form only, INSPECT? cannot be used to send a waveform back into the LSA1000. Users who require this capability or speed or both should instead use the WAVEFORM query or commands. It is possible to examine just a part of the waveform or a sparsed form of it, using the WAVEFORM_SETUP command covered later in this chapter.

Programmers might find it convenient, too, to combine the capabilities of the inspect facility with the waveform query command in order to construct files containing a plain text version of the waveform descriptor together with the full waveform in a format suitable for retransmission to the instrument. This can be done for a waveform in a memory location by sending the command

```
MC:INSPECT? “WAVEDESC”;WAVEFORM?
```

and putting the response directly into a disk file.

WAVEFORM?, Related Commands, and Blocks

Using the WAVEFORM? query is an effective way to transfer waveform data using the block formats defined in the IEEE-488.2 standard. Responses can then be downloaded back into the instrument using the WAVEFORM command.

All of a waveform's logical blocks can be read with the single query:

```
C1:WAVEFORM?
```

This is the preferred form for most applications due to its completeness. Time and space are the advantages when reading many waveforms with the same acquisition conditions, or when the interest is only in large amounts of raw integer data.

And any single block can be chosen for reading with a query such as:

```
C1:WAVEFORM? DAT1
```

The description In the *System Commands* section provides the various block names.

***Note:* A waveform query response can easily be a block containing over 16 million bytes if it is in binary format and twice as much if the HEX option is used.**

Interpreting the Waveform Descriptor

The binary response to a query of the form:

`C1:WAVEFORM?` or **`C1:WAVEFORM? ALL`**

can be placed in a disk file and then dumped to show the following hexadecimal and ASCII form:

```
➢ Byte          Binary Contents in                              ASCII
  Offset #        Hexadecimal                                   Translation
                                                                (.= uninteresting)

 0            43 31 3A 57 46 20 41 4C 4C 2C 23 39 30 30 30 30   C1:WFALL,#90000
16            30 30 34 35 30                                    00450

       0                     57 41 56 45 44 45 53 43 00 00 00        WAVEDESC...
 32   11      00 00 00 00 00 4C 45 43 52 4F 59 5F 32 5F 32 00   .....LECROY_2_2.
 48   27      00 00 00 00 00 00 01 00 00 00 00 01 5A 00 00 00   ................
 64   43      00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00   ................
 80   59      00 00 00 00 68 00 00 00 00 00 00 00 00 00 00 00   ................
 96   75      00 4C 45 43 52 4F 59 4C 53 41 31 30 30 30 00 00   .LECROYLSA1000..
112   91      00 37 84 09 40 00 00 00 00 00 00 00 00 00 00 00
128  107      00 00 00 00 00 00 00 00 34 00 00 00 34 00 00 00
144  123      32 00 00 00 00 00 00 00 33 00 00 00 00 00 00 00
160  139      01 00 00 00 00 00 00 00 01 00 00 00 01 00 00 00
176  155      00 34 83 12 6F 3A 0D 8E C9 46 FE 00 00 C7 00 00
192  171      00 00 08 00 01 32 2B CC 77 BE 6B A4 BB 51 A0 69
208  187      BB BE 6A D7 F2 A0 00 00 00 56 00 00 00 00 00 00
224  203      00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
240  219      00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
256  235      00 00 00 00 00 00 00 00 00 53 00 00 00 00 00 00
272  251      00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
288  267      00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
304  283      00 00 00 00 00 00 00 00 00 00 00 00 00 40 3B 00
320  299      00 00 00 00 00 17 0A 05 02 07 C8 00 00 00 00 00
336  315      00 00 00 00 00 00 00 00 01 00 0E 00 04 3F 80 00
352  331      00 00 0A 00 00 3F 80 00 00 3A 0D 8E C9 00 00
                                                            11
367         0 00 13 00 04 00 FA 00 09 00 16 00 0B 00 F3 00 E8
368         1 00 08 00 1B 00 1B 00 FA 00 EC 00 05 00 1B 00 1A
384        17 00 FC 00 EC 00 05 00 14 00 18 00 03 00 F8 00 0D
400        33 00 15 00 14 00 03 00 F4 00 05 00 11 00 10 00 F8
416        49 00 F0 00 11 00 1D 00 1E 00 F9 00 EA 00 06 00 1D
432        65 00 18 00 FE 00 EE 00 07 00 19 00 18 00 03 00 FA
448        81 00 0A 00 16 00 11 00
464        97

471  (Terminator)               0A
```

Here, in order to illustrate the contents of the logical blocks, the relevant parts (see explanations next page) have been separated. In addition, to facilitate counting, the corresponding Byte Offset numbering has been restarted each time a new block begins. The ASCII translation, only part of which is shown, has been similarly split and highlighted, showing how its parts correspond to the binary contents, highlighted in the same fashion.

On the preceding page… The first 10 bytes translate into ASCII and resemble the simple beginning of a query response. These are followed by the string **“#9000000450**”, the beginning of a binary block in which nine ASCII integers are used to give the length of the block (450 bytes). The waveform itself starts immediately after this, at Byte number 21. The very first byte is Byte #0, as it is for the first byte in each block (at the head of each of the three Byte Offset columns illustrated).

The first object is a DESCRIPTOR_NAME, a string of 16 characters with the value **WAVEDESC**.

Then, 16 bytes after the beginning of the descriptor (or at Byte #37, counting from the very start and referring to the numbers in the first Byte Offset column), we find the beginning of the next string: the TEMPLATE_NAME with the value **LECROY_2_2**.

Several other parameters follow. The INSTRUMENT_NAME, 76 bytes from the descriptor start (Byte #97), is easily recognizable. On the preceding line, at 38 bytes after the descriptor (Byte #59), a four-byte-long integer gives the length of the descriptor:

WAVE_DESCRIPTOR = 00 00 01 5A (hex) = 346.

At 60 bytes from the descriptor start (or Byte #81) we find another four-byte integer giving the length of the data array:

WAVE_ARRAY_1 = 00 00 00 68 (hex) = 104.

And at 116 bytes after the descriptor (Byte #137), yet another four-byte integer gives the number of data points:

WAVE_ARRAY_COUNT = 00 0000 34 (hex) = 52.

Now we know that the data will start at 346 bytes from the descriptor’s beginning (Byte #367), and that each of the 52 data points will be represented by two bytes. The waveform has a total length of 346 + 104, which is the same as the ASCII string indicated at the beginning of the block. The final 0A at Byte #471 is the NL character associated with the message terminator <NL><EOI>.

As the example was taken using an instrument with an eight-bit ADC, we see the eight bits followed by a 0 byte for each data point. However, for many other kinds of waveform this second byte will not be zero and will contain significant information. The data is coded in signed form (two's complement) with values ranging from –32768 = 8000 (hex) to 32767 = 7FFF (hex). If we had chosen to use the BYTE option for the data format the values would have been signed integers in the range –128 = 80 (hex) to 127 = 7F (hex). These ADC values are mapped to the display grid in the following way:

- 0 is located on the grid's center axis
- 127 (BYTE format) or 32767 (WORD format) is located at the top of the grid
- –128 (BYTE format) or –32768 (WORD format) is located at the bottom of the grid.

Interpreting Vertical Data

Now that we know how to decipher the data it would be useful to convert it to the appropriate measured values.

The vertical reading for each data point depends on the vertical gain and the vertical offset given in the descriptor. For acquisition waveforms this corresponds to the volts/div and voltage offset selected after conversion for the data representation being used. The template tells us that the vertical gain and offset can be found at bytes 156 and 160 respective of the descriptor start and that they are stored as floating point numbers in the IEEE 32-bit format. An ASCII string giving the vertical unit is to be found in VERTUNIT, Byte #196. The vertical value is given by the relationship:

value = VERTICAL_GAIN × data – VERTICAL_OFFSET

In the case of the data shown above we find:

VERTICAL_GAIN =	2.44141e–07 from the floating point number 3483 126f at byte 177
VERTICAL_OFFSET =	0.00054 from the floating point number 3A0D 8EC9 at byte 181
VERTICAL_UNIT =	V = volts from the string 5600 ... at byte 217

and therefore:

since data[4] =	FA00 = 64000 from the hexadecimal word FA00 at byte 371. Overflows the maximum. 16 bit value of 32767, so must be a negative value. Using the two's complement conversion $64000 – 2^{16} = –1536$
value[4] =	–0.000915 V as stated in the inspect command.

If the computer or the software available is not able to understand the IEEE floating point values, a description is to be found in the template.

The data values in a waveform may not all correspond to measured points. FIRST_VALID_PNT and LAST_VALID_PNT give the necessary information. The descriptor also records the SPARSING_FACTOR, the FIRST_POINT, and the SEGMENT_INDEX to aid interpretation if the options of the WAVEFORM_SETUP command have been used.

For waveforms such as the extrema and the complex FFT there will be two arrays — one after the other — for the two of the result.

Calculating a Data Point's Horizontal Position

Each vertical data value has a corresponding horizontal position, usually measured in time or frequency units. Every data value has a position, i, in the original waveform, with i = 0 corresponding to the first data point acquired. The descriptor parameter HORUNIT gives a string with the name of the horizontal unit.

Single-Sweep Waveforms

x[i] = HORIZ_INTERVAL × i + HORIZ_OFFSET

For acquisition waveforms this time is from the trigger to the data point in question. It will be different from acquisition to acquisition since the HORIZ_OFFSET is measured for each trigger.

In the case of the data shown above this means:

HORIZ_INTERVAL =	1e–08 from the floating point number 322b cc77 at byte 194

HORIZ_OFFSET =	–5.149e–08 from the double precision floating point number be6b a4bb 51a0 69bb at byte 198
HORUNIT = S =	seconds from the string 5300 ... at byte 262.

This gives:

x[0] = –5.149e–08 S
x[1] = –4.149e–08 S.

WAVEFORM Commands

Waveforms that have been read in their entirety with the WAVEFORM? query can be sent back into the instrument using WAVEFORM and other, related commands. Since the descriptor contains all of the necessary information, care need not be taken with any of the communication format parameters. The instrument can learn all it needs to know from the waveform.

***Note:* Waveforms can only be sent back to memory traces (M1, M2, M3, M4). This means possibly removing or changing the prefix (C1 or CHANNEL_1) in the response to the WF? query. See the System Commands for examples.**

When synthesizing waveforms for display or comparison, in order to ensure that the descriptor is coherent, read out a waveform of the appropriate size and then replace the data with the desired values.

There are many ways to use WAVEFORM and related commands to simplify or speed up work. Among them:

- **Partial Waveform Readout:** The WAVEFORM_SETUP command allows specification of a short part of a waveform for readout, as well as selection of a sparsing factor for reading only every n'th data point.
- **Byte Swapping:** The COMM_ORDER command allows the swapping of the two bytes of data presented in 16-bit word format (can be in the descriptor or in the the data/time arrays), when sending the data over the remote-control ports. This allows easier data interpretation, depending on the computer system used:
  - Intel-based computers — the data should be sent with the LSB first, and the command should be CORD LO.

  - ➢ Motorola-based computers — the data should be sent with the MSB first (CORD HI). This is the default at power-up.

- ➢ **Data Length, Block Format, and Encoding**: The COMM_FORMAT command gives control over these parameters. If the extra precision of the lower order byte of the standard data value is not needed, the BYTE option allows a saving of a factor of two on the amount of data to be transmitted or stored. If the computer being used is unable to read binary data, the HEX option allows a response form where the value of each byte is given by a pair of hexadecimal digits.

- ➢ **Data-Only Transfers**: The COMM_HEADER OFF mode enables a response to WF? DAT1 with the data only (the C1:WF DAT1 will disappear).

  If COMM_FORMAT OFF,BYTE,BIN has also been specified, the response will be mere data bytes (the #90000nnnnn will disappear).

High-Speed Waveform Transfer

Several important factors need to taken into account for achieving maximum continuous-data-transfer rates from server to client.

The single most important of these is the limiting of work done in the computer. This effectively means avoiding writing data to disk wherever possible, as well as minimizing operations such as per-data-point computations and reducing the number of calls to the IO system. Ways of doing this include:

- ➢ Reducing the number of points to be transferred and the number of data bytes per point. The pulse parameter capability and the processing functions can save a great deal of computing and a lot of data transfer time if employed creatively.
- ➢ Attempting to overlap waveform acquisition with waveform transfer. The LSA1000 is capable of transferring an already acquired or processed waveform after a new acquisition has been started. If the instrument is obliged to wait for triggers, overlapping waveform acquisition with waveform transfer will considerably increase the total time that the instrument will be able to acquire events (live time).

Example

The desirable type of command is:

`ARM; WAIT;C1:WF?`	to wait for the event, transfer the data, and then start a new acquisition.

This line can be “looped” in the program as soon as it has finished reading the waveform.

4 Waveform Structure

Using Status Registers

A wide range of status registers allows the LSA1000's internal processing status to be determined quickly at any time. These registers and the instrument's status reporting system are designed to comply with IEEE 488.2 recommendations. *Following an overview, starting this page, each of the registers and their roles are described.*

Related functions are grouped together in common status registers. Some, such as the Status Byte Register (STB) or the Standard Event Status Register (ESR), are required by the IEEE 488.2 Standard. Other registers are device-specific, and include the Command Error Register (CMR) and Execution Error Register (EXR). Those commands associated with IEEE 488.2 mandatory status registers are preceded by an asterisk <*>.

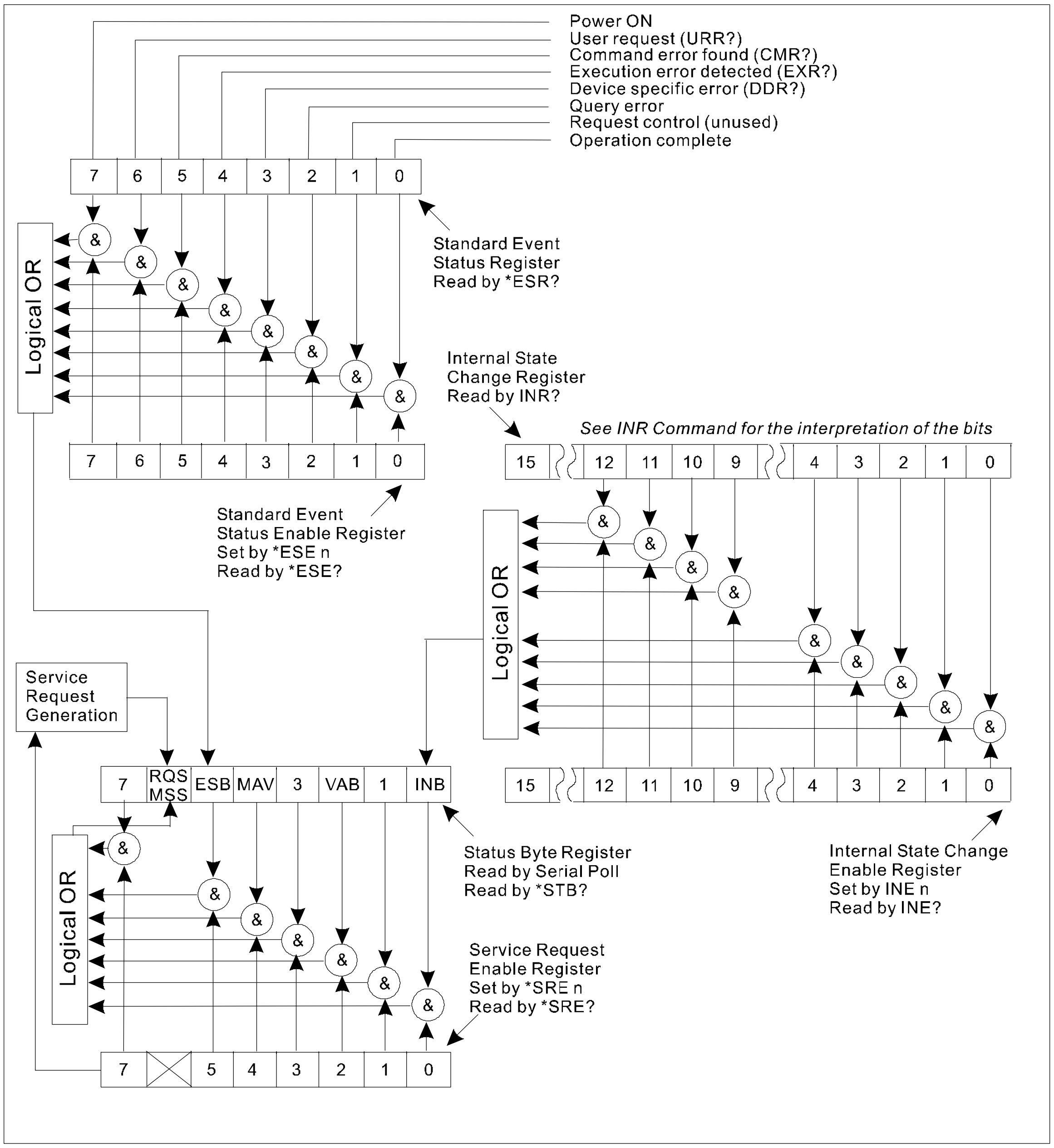
Overview

The Standard Event Status Bit (ESB) and the Internal Status Change Bit (INB) in the Status Byte Register are summary bits of the Standard Event Status Register (ESR) and the Internal State Change Register (INR). The Message Available Bit (MAV) is set whenever there are data bytes in the output queue. The Value Adapted Bit (VAB) indicates that a parameter value was adapted during a previous command interpretation (for example, if the command "TDIV 2.5 US" is received, the timebase is set to 2 µs/div along with the VAB bit).

The Master Summary Status bit (MSS) indicates a request for service from the instrument. The MSS bit can only be set if one or more of the other bits of STB are enabled with the Service Request Enable Register (SRE).

All Enable registers (SRE, ESE and INE) are used to generate a bit-wise AND with their associated status registers. The logical OR of this operation is reported to the STB register. At power-on, all Enable registers are zero, inhibiting any reporting to the STB.

The Standard Event Status Register (ESR) primarily summarizes errors, whereas the Internal State Change Register (INR) reports internal changes to the instrument. Additional details of errors reported by ESR can be obtained with the queries “CMR?”, “DDR?” and “EXR?”.



Power ON
User request (URR?)
Command error found (CMR?)
Execution error detected (EXR?)
Device specific error (DDR?)
Query error
Request control (unused)
Operation complete
7 6 5 4 3 2 1 0
Standard Event Status Register Read by *ESR?
Logical OR
&
Internal State Change Register Read by INR?
See INR Command for the interpretation of the bits
15 12 11 10 9 4 3 2 1 0
Standard Event Status Enable Register Set by *ESE n Read by *ESE?
Service Request Generation
7 RQS MSS ESB MAV 3 VAB 1 INB
Status Byte Register Read by Serial Poll Read by *STB?
Internal State Change Enable Register Set by INE n Read by INE?
Service Request Enable Register Set by *SRE n Read by *SRE?
7 5 4 3 2 1 0

Status Register Structure

The register structure contains one additional register, not shown in the figure on the previous page. This is the Parallel Poll Enable Register (PRE), which behaves exactly like the Service Request Enable Register (SRE), but sets the "ist" bit (also not shown in the figure), used in the Parallel Poll. The "ist" bit can also be read with the "*IST?" query.

Example

If an erroneous remote command — "TRIG_MAKE SINGLE", for example — is transmitted to the instrument, it rejects the command and sets the Command Error Register (CMR) to the value 1 (unrecognized command/query header). The non-zero value of CMR is reported to Bit 5 of the Standard Event Status Register (ESR), which is then set.

Nothing further occurs unless the corresponding Bit 5 of the Standard Event Status Enable Register (ESE) is set (with the command "*ESE 32"), enabling Bit 5 of ESR to be set for reporting to the summary bit ESB of the Status Byte Register (STB).

If setting of the ESB summary bit in STB is enabled, again nothing occurs unless further reporting is enabled by setting the corresponding bit in the Service Request Enable Register (with the command "*SRE 32"). In this case, the generation of a non-zero value of CMR ripples through to the Master Summary Status bit (MSS), generating a Service Request (SRQ).

The value of CMR can be read and simultaneously reset to zero at any time with the command "CMR?". The occurrence of a command error can also be detected by analyzing the response to "*ESR?". However, if several types of potential errors must be surveyed, it is usually far more efficient to enable propagation of the errors of interest into the STB with the enable registers ESE and INE.

Summary

A command error (CMR) sets Bit 5 of ESR if:

- Bit 5 of ESE is set, ESB of STB is also set, or
- Bit 5 of SRE is set, MSS/RQS of STB is also set and a Service Request is generated.

Status Byte Register (STB)

The Status Byte Register (STB) is the instrument’s central reporting structure. The STB is composed of eight single-bit summary messages (of which three are unused), which reflect the current status of the associated data structures implemented in the instrument:

- **Bit 0** is the summary bit INB of the Internal State Change Register. It is set if any of the bits of the INR are set, provided they are enabled by the corresponding bit of the INE register.
- **Bit 2** is the Value Adapted bit, indicating that a parameter value was adapted during a previous command interpretation.
- **Bit 4** is the Message Available (MAV) bit, indicating that the interface output queue is not empty.
- **Bit 5** is the summary bit ESB of the Standard Event Status Register. It is set if any of the bits of the ESR are set, provided they are enabled by the corresponding bit of the ESE register.
- **Bit 6** is either the Master Summary Status bit (MSS) or the Request for Service bit (RQS), owing to the STB being able to be read in two different ways. The command “*STB?” reads and clears the STB in the query mode, in which case Bit 6 is the MSS bit, and indicates whether the instrument has any reason for requesting service.

The Status Byte Register can be read using the query “*STB?”. The response represents the binary weighted sum of the register bits. The register is cleared by “*STB?”, “ALST?”, “*CLS”, or after the instrument has been powered up.

Another way of reading the STB is using the serial poll (*see “Instrument Polls”, Chapter 3*). In this case, Bit 6 is the RQS bit, indicating that the instrument has activated the SRQ line on the GPIB. The serial poll only clears the RQS bit. Therefore, the MSS bit of the STB (and any other bits which caused MSS to be set) will stay set after a serial poll. These bits must be reset.

Standard Event Status Register (ESR)

The Standard Event Status Register (ESR) is a 16-bit register reflecting the occurrence of events. The ESR bit assignments have been standardized by IEEE 488.2. Only the lower eight bits are currently in use.

The ESR is read using the query “*ESR?”. The response is the binary weighted sum of the register bits. The register is cleared with an “*ESR?” or “ALST?” query, a “*CLS” command or after power-on.

Example

The response message “*ESR 160” indicates that a command error occurred and that the ESR is being read for the first time after power-on. The value 160 can be broken down into 128 (Bit 7) plus 32 (bit 5). *See the table on the same page as the ESR command description for the conditions corresponding to the bits set.*

The “Power ON” bit appears only on the first “*ESR?” query after power-on because the query clears the register. This type of command error can be determined by reading the Command Error Status Register with the query “CMR?”. Note that it is not necessary to read (nor simultaneously clear) this register in order to set the CMR bit in the ESR on the next command error.

Standard Event Status Enable Register (ESE)

The Standard Event Status Enable Register (ESE) allows one or more events in the Standard Event Status Register to be reported to the ESB summary bit in the STB.

The ESE is modified with the command “*ESE” and cleared with the command “*ESE 0”, or after power-on. It is read with the query “*ESE?”.

Example

“*ESE 4” sets bit 2 (binary 4) of the ESE Register, enabling query errors to be reported.

Service Request Enable Register (SRE)

The Service Request Enable Register (SRE) specifies which summary bit(s) in the Status Byte Register will bring about a service request. The SRE consists of eight bits. Setting a bit in this register allows the summary bit located at the same bit position in the Status Byte Register to generate a service request, provided that the associated event becomes true. Bit 6 (MSS) cannot be set and is always reported as zero in response to the query “*SRE?”.

SRE is modified with the command "*SRE" and cleared with the command "*SRE 0", or after power-on. It may be read with the query "*SRE?".

Parallel Poll Enable Register (PRE)

The Parallel Poll Enable Register (PRE) specifies which summary bit(s) in the Status Byte Register will set the "ist" individual local message. This register is quite similar to the Service Request Enable Register (SRE), but is used to set the parallel poll "ist" bit rather than MSS.

The value of the "ist" may also be read without a Parallel Poll via the query "*IST?". The response indicates whether or not the "ist" message has been set (values are 1 or 0).

The PRE is modified with the command "*PRE" and cleared with the command "*PRE 0", or after power-on. It is read with the query "*PRE?". (*See Chapter 3 "Instrument Polls".*)

Example

"*PRE 5" sets bits 2 and 0 (decimal 4 and 1) of the Parallel Poll Enable Register.

Internal State Change Status Register (INR)

The Internal State Change Status Register (INR) reports the completion of a number of internal operations (*the events tracked by this 16-bit-wide register are listed with the "INR?" query in the System Commands section*).

The INR is read using the query "INR?". The response is the binary-weighted sum of the register bits. The register is cleared with an "INR?" or "ALST?" query, a "*CLS" command, or after power-on.

Internal State Change Enable Register (INE)

The Internal State Change (INE) allows one or more events in the Internal State Change Status Register to be reported to the INB summary bit in the STB.

The INE is modified with the command "INE" and cleared with the command "INE 0", or after power-on. It is read with the query "INE?".

Command Error Status Register (CMR)

The Command Error Status register contains the code of the last command error detected by the instrument. Command error codes are listed with the command "CMR?".

The Command Error Status Register may be read using the query "CMR?". The response is the error code. The register is cleared with a "CMR?" or "ALST?" query, a "*CLS" command, or after power-on.

Device Dependent Error Status Register (DDR)

The Device Dependent Error Status Register (DDR) indicates the type of hardware errors affecting the instrument. Individual bits in this register report specific hardware failures. They are listed with the command “DDR?”.

The DDR is read using the “DDR?” query. The response is the binary weighted sum of the error bits. The register is cleared with a “DDR?” or “ALST?” query, a “*CLS” command, or after power-on.

Execution Error Status Register (EXR)

The Execution Error Status Register (EXR) contains the code of the last execution error detected by the instrument. Execution error codes are listed with the command “EXR?”.

The EXR is read using the “EXR?” query. The response is the error code. The register is cleared with an “EXR?” or “ALST?” query, a “*CLS” command, or after power-on.

5 Status Registers

About These Commands & Queries

This section lists and describes the remote control commands and queries recognized by the instrument. All commands and queries can be executed in either local or remote state. Where not included here, those for special options can be found in the options' dedicated Operator's Manuals.

The description for each command or query, with syntax and other information, begins on a new page. The name (header) is given in both long and short form at the top of the page, and the subject is indicated as a command or query or both. Queries perform actions such as obtaining information, and are recognized by the question mark (?) following the header.

How They are Listed

The descriptions are listed in alphabetical order according to their *long* form. Thus the description of ATTENUATION, whose short form is ATTN, is listed before that of AUTO_CALIBRATE, whose short form is ACAL. The two special indexes at the beginning of this section (*pages 3 to 8*) are designed as reference aids for quickly finding commands and queries. One lists the commands and queries in alphabetical order according to short form, while the other groups them according to subsystem or category.

How They are Described

In the descriptions themselves, a brief explanation of the function performed is given. This is followed by a presentation of the formal syntax, with the header given in **`Upper-and-Lower-Case`** characters and the short form derived from it in **`ALL UPPER-CASE`** characters. Where applicable, the syntax of the query is given with the format of its response.

A short example illustrating a typical use is also presented.

When Can They be Used?

All the commands and queries listed here can be used with the standard LSA1000, except when a particular option is required. The raised hand symbol ☝ indicates a note on availability for a particular option or function.

Command Notation

The following notation is used in the commands:

< >	Angular brackets enclose words that are used as placeholders, of which there are two types: the header path and the data parameter of a command.
: =	A colon followed by an equals sign separates a placeholder from the description of the type and range of values that may be used in a command instead of the placeholder.
{ }	Braces enclose a list of choices, one of which one must be made.
[ ]	Square brackets enclose optional items.
...	An ellipsis indicates that the items both to its left and right may be repeated a number of times.

As an example, consider the syntax notation for the command to set the vertical input sensitivity:

<channel> **: VOLT_DIV** <v_gain>

<channel> : = {**C1**, **C2**}

<v_gain> : = 5.0 mV to 2.5 V

The first line shows the formal appearance of the command, with <channel> denoting the placeholder for the header path and <v_gain> the placeholder for the data parameter specifying the desired vertical gain value. The second line indicates that either **C1** or **C2** must be chosen for the header path. And the third explains that the actual vertical gain can be set to any value between 5 mV and 2.5 V.

Refer to Chapter 2 for an overview of the command functions and notation used.

Command Execution

Before attempting to execute a command or query, the LSA1000 scans it to verify its correctness and that sufficient information is given to perform the requested action.

Since interrogating the LSA1000 does not change its internal state, it may be queried at any time. The only exceptions to this are the queries *CAL? and *TST?, which both recalibrate the instrument.

Commands & Queries Tabled By Short Form

Page No.	Short Form	Long Form	Subsystem (category)	What the Command/Query Does
12	**ACAL**	**AUTO_CALIBRATE**	MISCELLANEOUS	Enables or disables automatic calibration.
10	**ALST?**	**ALL_STATUS?**	STATUS	Reads and clears the contents of all status registers.
9	**AOUT**	**ACQ_OUT**	ACQUISITION	Sets the mode of the ACQ OUT signal
11	**ARM**	**ARM_ACQUISITION**	ACQUISITION	Changes acquisition state from “stopped” to “single”.
13	**BWL**	**BANDWIDTH_LIMIT**	ACQUISITION	Enables/disables the bandwidth-limiting low-pass filter.
14	***CAL?**	***CAL?**	MISCELLANEOUS	Performs complete internal calibration of the instrument.
27	**CFMT**	**COMM_FORMAT**	COMMUNICATION	Selects the format for sending waveform data.
29	**CHDR**	**COMM_HEADER**	COMMUNICATION	Controls formatting of query responses.
30	**CHLP**	**COMM_HELP**	COMMUNICATION	Controls operational level of the RC Assistant.
31	**CHL?**	**COMM_HELP_LOG?**	COMMUNICATION	Returns the contents of the RC Assistant log.
17	**CLM**	**CLEAR_MEMORY**	FUNCTION	Clears the specified memory.
19	***CLS**	***CLS**	STATUS	Clears all status data registers.
18	**CLSW**	**CLEAR_SWEEPS**	FUNCTION	Restarts the cumulative processing functions.
15	**CMGN?**	**CAL_MARGIN?**	MISCELLANEOUS	Checks oil light margins and names margins and lights.
20	**CMR?**	**CMR?**	STATUS	Reads and clears the Command error Register (CMR).
22	**COLR**	**COLOR**	DISPLAY	Selects color of individual on-screen objects
25	**COMB**	**COMBINE_CHANNELS**	ACQUISITION	Controls the channel interleaving function.
26	**COMS**	**COMBINE_SOURCE**	ACQUISITION	Determines the input channel used for interleaving.
32	**CONET**	**COMM_NET**	COMMUNICATION	Specifies the LSA1000’s network address.
33	**CORD**	**COMM_ORDER**	COMMUNICATION	Controls the byte order of waveform data transfers.
34	**CRMS**	**CURSOR_MEASURE**	CURSOR	Specifies the type of cursor/parameter measurement.
37	**CRST**	**CURSOR_SET**	CURSOR	Allows positioning of any one of eight cursors.
39	**CRVA?**	**CURSOR_VALUE?**	CURSOR	Returns trace values measured by specified cursors.
24	**CSCH**	**COLOR_SCHEME**	DISPLAY	Selects the display color scheme.
16	**CSTS?**	**CAL_STATUS?**	MISCELLANEOUS	Checks internal control margins, gives calibration status.
41	**DATE**	**DATE**	MISCELLANEOUS	Changes the date/time of the internal real-time clock.
42	**DDR?**	**DDR?**	STATUS	Reads, clears the Device Dependent Register (DDR).
43	**DEF**	**DEFINE**	FUNCTION	Specifies math expression for function evaluation.
49	**DELF**	**DELETE_FILE**	MASS STORAGE	Deletes files from mass storage.
50	**DIR**	**DIRECTORY**	MASS STORAGE	Creates and deletes file directories.
52	**DISP**	**DISPLAY**	DISPLAY	Controls the display screen..
40	**DPNT**	**DATA_POINTS**	DISPLAY	Controls bold/single pixel display of sample points.
53	**DTJN**	**DOT_JOIN**	DISPLAY	Controls the interpolation lines between data points.
54	**DZOM**	**DUAL_ZOOM**	DISPLAY	Sets horizontal magnification and positioning.
55	***ESE**	***ESE**	STATUS	Sets the Standard Event Status Enable register(ESE).
56	***ESR?**	***ESR?**	STATUS	Reads, clears the Event Status Register (ESR).
59	**EXR?**	**EXR?**	STATUS	Reads, clears the EXecution error Register (EXR).
62	**FCR**	**FIND_CTR_RANGE**	FUNCTION	Sets histogram center and width.
63	**FCRD**	**FORMAT_CARD**	MASS STORAGE	Formats a memory card.
65	**FFLP**	**FORMAT_FLOPPY**	MASS STORAGE	Formats a floppy disk.
67	**FHDD**	**FORMAT_HDD**	MASS STORAGE	Formats a removable hard disk.
61	**FLNM**	**FILENAME**	MASS STORAGE	Changes default filenames.
70	**FRST**	**FUNCTION_RESET**	FUNCTION	Resets a waveform-processing function.

Page No.	Short Form	Long Form	Subsystem (category)	What the Command/Query Does
69	**FSCR**	**FULL_SCREEN**	DISPLAY	Selects magnified view format for the grid.
71	**GMOD**	**GAIN_MODE**	ACQUISITION	Specifies the gain mode (gain range) of the front end.
72	**GBWL**	**GLOBAL_BWL**	ACQUISITION	Enables/disables global bandwidth-limiting.
73	**GRID**	**GRID**	DISPLAY	Specifies single-, dual- or quad-mode grid display.
74	**HMAG**	**HOR_MAGNIFY**	DISPLAY	Horizontally expands the selected math trace.
75	**HPOS**	**HOR_POSITION**	DISPLAY	Horizontally positions intensified zone's center.
77	***IDN?**	***IDN?**	MISCELLANEOUS	For identification purposes.
78	**INE**	**INE**	STATUS	Sets the Internal state change Enable register (INE).
79	**INR?**	**INR?**	STATUS	Reads, clears INternal state change Register (INR).
82	**INSP?**	**INSPECT?**	WAVEFORM TRANS.	Allows acquired waveform parts to be read.
81	**INTS**	**INTENSITY**	DISPLAY	Sets the grid or trace/text intensity level.
84	***IST?**	***IST?**	STATUS	Reads the current state of the IEEE 488.
85	**MSIZ**	**MEMORY_SIZE**	ACQUISITION	Selects max. memory length.
86	**MZOM**	**MULTI_ZOOM**	DISPLAY	Sets horizontal magnification and positioning.
87	**OFST**	**OFFSET**	ACQUISITION	Allows output channel vertical offset adjustment.
88	***OPC**	***OPC**	STATUS	Sets the OPC bit in the Event Status Register (ESR).
89	***OPT?**	***OPT?**	MISCELLANEOUS	Identifies LSA1000 options.
91	**PACL**	**PARAMETER_CLR**	CURSOR	Clears all current parameters in Custom, Pass/Fail.
92	**PACU**	**PARAMETER_CUSTOM**	CURSOR	Controls parameters with customizable qualifiers.
96	**PADL**	**PARAMETER_DELETE**	CURSOR	Deletes a specified parameter in Custom, Pass/Fail.
97	**PAST?**	**PARAMETER_STATISTICS?**	CURSOR	Returns current statistics parameter values.
98	**PAVA?**	**PARAMETER_VALUE?**	CURSOR	Returns current parameter, mask test values.
108	**PECS**	**PER_CURSOR_SET**	CURSOR	Positions independent cursors.
110	**PECV?**	**PER_CURSOR_VALUE?**	CURSOR	Returns values measured by cursors.
113	**PELT**	**PERSIST_LAST**	DISPLAY	Shows the last trace drawn in a persistence data map.
111	**PERS**	**PERSIST**	DISPLAY	Enables or disables the persistence display mode.
112	**PECL**	**PERSIST_COLOR**	DISPLAY	Controls color rendering method of persistence traces.
114	**PESA**	**PERSIST_SAT**	DISPLAY	Sets the color saturation level in persistence.
115	**PESU**	**PERSIST_SETUP**	DISPLAY	Selects display persistence duration.
101	**PFCO**	**PASS_FAIL_CONDITION**	CURSOR	Adds a Pass/Fail test condition or custom parameter.
103	**PFCT**	**PASS_FAIL_COUNTER**	CURSOR	Resets the Pass/Fail acquisition counters.
104	**PFDO**	**PASS_FAIL_DO**	CURSOR	Defines desired outcome, actions after Pass/Fail test.
106	**PFMS**	**PASS_FAIL_MASK**	CURSOR	Generates tolerance mask on a trace and stores it.
107	**PFST?**	**PASS_FAIL_STATUS?**	CURSOR	Returns the Pass/Fail test for a given line number.
116	***PRE**	***PRE**	STATUS	Sets the PaRallel poll Enable register (PRE).
118	**RCLK**	**REFERENCE_CLOCK**	ACQUISITION	Selects the system clock source.
117	**REC**	**RECALL**	WAVEFORM TRANS.	Recalls a file from mass storage to internal memory.
119	***RST**	***RST**	SAVE/RECALL	The *RST command initiates a device reset.
120	**SEQ**	**SEQUENCE**	ACQUISITION	Sets the conditions for the sequence mode acquisition.
122	***SRE**	***SRE**	STATUS	Sets the Service Request Enable register (SRE).
123	***STB?**	***STB?**	STATUS	Reads the contents of the IEEE 488.
126	**STO**	**STORE**	WAVEFORM TRANS.	Stores a trace in internal memory or mass storage.
125	**STOP**	**STOP**	ACQUISITION	Immediately stops signal acquisition.
127	**STST**	**STORE_SETUP**	WAVEFORM TRANS.	Controls the way in which traces are stored.
128	**STTM**	**STORE_TEMPLATE**	WAVEFORM TRANS.	Stores the waveform template to mass storage.

Page No.	Short Form	Long Form	Subsystem (category)	What the Command/Query Does
130	TDIV	TIME_DIV	ACQUISITION	Modifies the timebase setting.
129	TMPL?	TEMPLATE?	WAVEFORM TRANS.	Produces a complete waveform template copy.
131	TRA	TRACE	DISPLAY	Enables or disables the display of a trace.
133	TRDL	TRIG_DELAY	ACQUISITION	Sets the time at which the trigger is to occur.
132	*TRG	*TRG	ACQUISITION	Executes an ARM command.
134	TRLV	TRIG_LEVEL	ACQUISITION	Adjusts the trigger level of the specified trigger source.
135	TRMD	TRIG_MODE	ACQUISITION	Specifies the trigger mode.
136	TRSE	TRIG_SELECT	ACQUISITION	Selects the condition that will trigger acquisition.
137	TRSL	TRIG_SLOPE	ACQUISITION	Sets the trigger slope of the specified trigger source.
138	TRWI	TRIG_WINDOW	ACQUISITION	Sets window amplitude on current Edge trigger source.
139	*TST?	*TST?	MISCELLANEOUS	Performs an internal self-test.
142	VDIV	VOLT_DIV	ACQUISITION	Sets the vertical sensitivity.
140	VMAG	VERT_MAGNIFY	DISPLAY	Vertically expands the specified trace.
141	VPOS	VERT_POSITION	DISPLAY	Adjusts the vertical position of the specified trace.
143	VRNG	VOLT_RANGE	ACQUISITION	Sets the full-scale range in volts.
144	*WAI	*WAI	STATUS	Required by the IEEE 488.
145	WAIT	WAIT	ACQUISITION	Prevents new analysis until current is completed.
146	WF	WAVEFORM	WAVEFORM TRANS.	Transfers a waveform from controller to LSA1000.
148	WFSU	WAVEFORM_SETUP	WAVEFORM TRANS.	Specifies amount of waveform data to go to controller.
150	WFTX	WAVEFORM_TEXT	WAVEFORM TRANS.	Documents acquisition conditions.
151	XYAS?	XY_ASSIGN?	DISPLAY	Returns traces currently assigned to the XY display.
152	XYCO	XY_CURSOR_ORIGIN	CURSOR	Sets origin position of absolute cursor measurements.
153	XYCS	XY_CURSOR_SET	CURSOR	Allows positioning of XY voltage cursors.
155	XYCV?	XY_CURSOR_VALUE?	CURSOR	Returns the current values of the X vs Y cursors.
157	XYDS	XY_DISPLAY	DISPLAY	Enables or disables the XY display mode.
158	XYSA	XY_SATURATION	DISPLAY	Sets persistence color saturation level in XY display.

Commands & Queries Tabled By Subsystem

ACQUISITION — Controlling Waveform Acquisition

9	**AOUT**	**ACQ_OUT**	Sets the mode of the ACQ OUT signal.
11	**ARM**	**ARM_ACQUISITION**	Changes acquisition state from “stopped” to “single”.
13	**BWL**	**BANDWIDTH_LIMIT**	Enables or disables the bandwidth-limiting low-pass filter.
26	**COMS**	**COMBINE_SOURCE**	Determines the input channel used for interleaving.
25	**COMB**	**COMBINE_CHANNELS**	Controls the acquisition system’s channel-interleaving function.
72	**GBWL**	**GLOBAL_BWL**	Enables/disables global bandwidth-limiting.
71	**GMOD**	**GAIN_MODE**	Specifies the gain mode (gain range) of the front end.
85	**MSIZ**	**MEMORY_SIZE**	Allows selection of maximum memory length (M- and L-models only).
87	**OFST**	**OFFSET**	Allows vertical offset adjustment of the specified input channel.
118	**RCLK**	**REFERENCE_CLOCK**	Selects the system clock source.
120	**SEQ**	**SEQUENCE**	Sets the conditions for the sequence mode acquisition.
125	**STOP**	**STOP**	Immediately stops signal acquisition.
130	**TDIV**	**TIME_DIV**	Modifies the timebase setting.
133	**TRDL**	**TRIG_DELAY**	Sets the time at which the trigger is to occur.
132	***TRG**	***TRG**	Executes an ARM command.
134	**TRLV**	**TRIG_LEVEL**	Adjusts the level of the specified trigger source.
135	**TRMD**	**TRIG_MODE**	Specifies Trigger mode.
136	**TRSE**	**TRIG_SELECT**	Selects the condition that will trigger acquisition.
137	**TRSL**	**TRIG_SLOPE**	Sets the slope of the specified trigger source.
138	**TRWI**	**TRIG_WINDOW**	Sets the window amplitude in volts on the current Edge trigger source.
142	**VDIV**	**VOLT_DIV**	Sets the vertical sensitivity in volts/div.
143	**VRNG**	**VOLT_RANGE**	Sets the full-scale range in volts.
145	**WAIT**	**WAIT**	Prevents new command analysis until current acquisition completion.

COMMUNICATION — Setting Communication Characteristics

27	**CFMT**	**COMM_FORMAT**	Selects the format to be used for sending waveform data.
29	**CHDR**	**COMM_HEADER**	Controls formatting of query responses.
30	**CHLP**	**COMM_HELP**	Controls operational level of the RC Assistant.
31	**CHL?**	**COMM_HELP_LOG?**	Returns the contents of the RC Assistant log.
32	**CONET**	**COMM_NET**	Specifies the LSA1000’s network address.
33	**CORD**	**COMM_ORDER**	Controls the byte order of waveform data transfers.

CURSOR — Performing Measurements

34	**CRMS**	**CURSOR_MEASURE**	Specifies the type of cursor or parameter measurement for display.
37	**CRST?**	**CURSOR_SET?**	Allows positioning of any one of eight cursors.
39	**CRVA?**	**CURSOR_VALUE?**	Returns the values measured by the specified cursors for a given trace.
91	**PACL**	**PARAMETER_CLR**	Clears all current parameters in Custom and Pass/Fail modes.
92	**PACU**	**PARAMETER_CUSTOM**	Controls parameters with customizable qualifiers.
96	**PADL**	**PARAMETER_DELETE**	Deletes a specified parameter in Custom and Pass/Fail modes.
97	**PAST?**	**PARAMETER_STATISTICS?**	Returns current statistics values for the specified pulse parameter.
98	**PAVA?**	**PARAMETER_VALUE?**	Returns current value(s) of parameter(s) and mask tests.
108	**PECS**	**PER_CURSOR_SET**	Allows positioning of any one of six independent cursors.
110	**PECV?**	**PER_CURSOR_VALUE?**	Returns the values measured by specified cursors for a given trace.
101	**PFCO**	**PASS_FAIL_CONDITION**	Adds a Pass/Fail test condition or custom parameter to display.

103	**PFCT**	**PASS_FAIL_COUNTER**	Resets the Pass/Fail acquisition counters.
104	**PFDO**	**PASS_FAIL_DO**	Defines the desired outcome and actions following a Pass/Fail test.
106	**PFMS**	**PASS_FAIL_MASK**	Generates a tolerance mask around a chosen trace and stores it.
107	**PFST?**	**PASS_FAIL_STATUS?**	Returns the Pass/Fail test for a given line number.
152	**XYCO**	**XY_CURSOR_ORIGIN**	Sets position of origin for absolute cursor measurements on XY display.
153	**XYCS**	**XY_CURSOR_SET**	Allows positioning of any one of six independent XY voltage cursors.
155	**XYCV?**	**XY_CURSOR_VALUE?**	Returns current values of X vs Y cursors.

DISPLAY — Displaying Waveforms

22	**COLR**	**COLOR**	Selects color of individual objects: traces, grids or cursors.
24	**CSCH**	**COLOR_SCHEME**	Selects the display color scheme..
40	**DPNT**	**DATA_POINTS**	Controls display of sample points in single display pixels or bold.
52	**DISP**	**DISPLAY**	Controls the oscilloscope display screen.
53	**DTJN**	**DOT_JOIN**	Controls the interpolation lines between data points.
54	**DZOM**	**DUAL_ZOOM**	Sets horiz. magnification and positioning for all expanded traces.
69	**FSCR**	**FULL_SCREEN**	Selects magnified view format for the grid.
73	**GRID**	**GRID**	Specifies grid display in single, dual or quad mode.
74	**HMAG**	**HOR_MAGNIFY**	Horizontally expands selected expansion trace.
75	**HPOS**	**HOR_POSITION**	Horizontally positions intensified zone's center on source trace.
81	**INTS**	**INTENSITY**	Sets grid or trace/text intensity level.
86	**MZOM**	**MULTI_ZOOM**	Sets horiz. magnification and positioning for all expanded traces.
111	**PERS**	**PERSIST**	Enables or disables the Persistence Display mode.
112	**PECL**	**PERSIST_COLOR**	Controls color rendering method of persistence traces.
113	**PELT**	**PERSIST_LAST**	Shows the last trace drawn in a persistence data map.
114	**PESA**	**PERSIST_SAT**	Sets the color saturation level in persistence.
115	**PESU**	**PERSIST_SETUP**	Selects display persistence duration in Persistence mode.
131	**TRA**	**TRACE**	Enables or disables the display of a trace.
140	**VMAG**	**VERT_MAGNIFY**	Vertically expands the specified trace.
141	**VPOS**	**VERT_POSITION**	Adjusts the vertical position of the specified trace.
151	**XYAS?**	**XY_ASSIGN?**	Returns the traces currently assigned to the XY display.
157	**XYDS**	**XY_DISPLAY**	Enables or disables the XY display mode.
158	**XYSA**	**XY_SATURATION**	Sets persistence color saturation level in XY display.

FUNCTION — Performing Waveform Mathematical Operations

17	**CLM**	**CLEAR_MEMORY**	Clears the specified memory.
18	**CLSW**	**CLEAR_SWEEPS**	Restarts the cumulative processing functions.
43	**DEF**	**DEFINE**	Specifies the mathematical expression to be evaluated by a function.
62	**FCR**	**FIND_CONTROL_RANGE**	Sets histogram center and width.
70	**FRST**	**FUNCTION_RESET**	Resets a waveform processing function.

MASS STORAGE — Creating and Deleting File Directories

49	**DELF**	**DELETE_FILE**	Deletes files from mass storage.
50	**DIR**	**DIRECTORY**	Creates and deletes file directories.
63	**FCRD**	**FORMAT_CARD**	Formats a memory card.
65	**FFLP**	**FORMAT_FLOPPY**	Formats a floppy disk.
67	**FHDD**	**FORMAT_HDD**	Formats a removable hard disk.
61	**FLNM**	**FILENAME**	Changes default filenames.

MISCELLANEOUS — Calibration and Testing

12	**ACAL**	**AUTO_CALIBRATE**	Enables or disables automatic calibration.
14	***CAL?**	***CAL?**	Performs a complete internal calibration of the instrument.
15	**CMGN?**	**CAL_MARGIN?**	Checks oil light margins and names margins and lights.
16	**CSTS?**	**CAL_STATUS?**	Checks internal control margins, gives last calibration status.
41	**DATE**	**DATE**	Changes the date/time of the LSA1000's internal real-time clock.
77	***IDN?**	***IDN?**	Used for identification purposes.
89	***OPT?**	***OPT?**	Identifies LSA1000 options.
139	***TST?**	***TST?**	Performs an internal self-test.

SAVE / RECALL SETUP — Preserving and Restoring Settings

119	***RST**	***RST**	The *RST command initiates a device reset.

STATUS — Obtaining Status Information and Setting Up Service Requests

10	**ALST?**	**ALL_STATUS?**	Reads and clears the contents of all (but one) of the status registers.
19	***CLS**	***CLS**	Clears all the status data registers.
20	**CMR?**	**CMR?**	Reads and clears the contents of the CoMmand error Register (CMR).
42	**DDR?**	**DDR?**	Reads and clears the Device-Dependent error Register (DDR).
55	***ESE**	***ESE**	Sets the standard Event Status Enable (ESE) register.
56	***ESR?**	***ESR?**	Reads and clears the Event Status Register (ESR).
59	**EXR?**	**EXR?**	Reads and clears the EXecution error Register (EXR).
78	**INE**	**INE**	Sets the INternal state change Enable register (INE).
79	**INR?**	**INR?**	Reads and clears the INternal state change Register (INR).
84	***IST?**	***IST?**	Individual STatus reads the current state of IEEE 488.
88	***OPC**	***OPC**	Sets to true the OPC bit (0) in the Event Status Register (ESR).
116	***PRE**	***PRE**	Sets the PaRallel poll Enable register (PRE).
122	***SRE**	***SRE**	Sets the Service Request Enable register (SRE).
123	***STB?**	***STB?**	Reads the contents of IEEE 488.
144	***WAI**	***WAI**	WAIt to continue — required by IEEE 488.

WAVEFORM TRANSFER — Preserving and Restoring Waveforms

82	**INSP?**	**INSPECT?**	Allows acquired waveform parts to be read.
117	**REC**	**RECALL**	Recalls a waveform file from mass storage to internal memories M1–4.
126	**STO**	**STORE**	Stores a trace in one of the internal memories M1–4 or mass storage.
127	**STST**	**STORE_SETUP**	Controls the way in which traces are stored.
128	**STTM**	**STORE_TEMPLATE**	Stores the waveform template in a mass-storage device.
129	**TMPL?**	**TEMPLATE?**	Produces a copy of the template describing a complete waveform.
146	**WF**	**WAVEFORM**	Transfers a waveform from the controller to the LSA1000.
148	**WFSU**	**WAVEFORM_SETUP**	Specifies amount of waveform data for transmission to controller.
150	**WFTX**	**WAVEFORM_TEXT**	Documents the conditions under which a waveform has been acquired.

ACQUISITION ACQ_OUT, AOUT

Command/Query

DESCRIPTION

The ACQ_OUT command specifies the operation of the ACQ OUT signal. In STD mode, the ACQ OUT signal is a hardware-generated pulse that occurs at the end of every acquisition segment. In SEQ mode, behavior is identical to STD except in sequence mode. In sequence mode, the pulse is software-generated at the end of a sequence acquisition only (i.e. after the acquisition of the last segment only).

COMMAND SYNTAX

```
Acq_OUT <mode>
<mode> := { STD, SEQ }
```

QUERY SYNTAX

```
Acq_OUT?
```

RESPONSE FORMAT

`Acq_OUT` <mode>

STATUS — ALL_STATUS?, ALST?

Query

DESCRIPTION

The ALL_STATUS? query reads and clears the contents of all status registers: STB, ESR, INR, DDR, CMR, EXR and URR except for the MAV bit (bit 6) of the STB register. For an interpretation of the contents of each register, refer to the appropriate status register.

The ALL_STATUS? query is useful in a complete overview of the state of the instrument.

QUERY SYNTAX

ALl_STatus?

RESPONSE FORMAT

ALl_STatus STB,<value>**,ESR,**<value>**,INR,**<value>, **DDR,**<value>**,CMR,**<value>**,EXR,**<value>**,URR,<**value>

<value> : = 0 to 65535

EXAMPLE

The following instruction reads the contents of all the status registers:

ALST?

Response message:

```
ALST TB,000000,ESR,000052,INR,000005,DDR,000000,
CMR,000004,EXR,000024,URR,000000
```

RELATED COMMANDS

*CLS, CMR?, DDR?, *ESR?, EXR?, *STB?, URR?

ACQUISITION — ARM_ACQUISITION, ARM
Command

DESCRIPTION

The ARM_ACQUISITION command enables the signal acquisition process by changing the acquisition state (trigger mode) from “stopped” to “single”.

COMMAND SYNTAX

```
ARM_acquisition
```

EXAMPLE

The following command enables signal acquisition:

```
ARM
```

RELATED COMMANDS

STOP, *TRG, TRIG_MODE, WAIT

MISCELLANEOUS AUTO_CALIBRATE, ACAL

Command/Query

DESCRIPTION

The AUTO_CALIBRATE command is used to enable or disable the automatic calibration of the instrument. At power-up, auto-calibration is turned ON, i.e. all input channels are periodically calibrated for the current input amplifier and timebase settings.

The automatic calibration may be disabled by issuing the command ACAL OFF. Whenever it is convenient, a *CAL? query may be issued to fully calibrate the LSA1000. When the LSA1000 is returned to local control, the periodic calibrations are resumed.

The response to the AUTO_CALIBRATE? query indicates whether auto-calibration is enabled.

COMMAND SYNTAX

`Auto_CALibrate` <state>

<state> : = {**`ON`**, **`OFF`**}

QUERY SYNTAX

`Auto_CALibrate?`

RESPONSE FORMAT

`Auto_CALibrate` <state>

EXAMPLE

The following instruction disables auto-calibration:

```
ACAL OFF
```

RELATED COMMANDS

*CAL?, CAL_STATUS

ACQUISITION BANDWIDTH_LIMIT, BWL

Command/Query

DESCRIPTION

BANDWIDTH_LIMIT enables or disables the bandwidth-limiting low-pass filter. When Global_BWL (*see page 72*) is on the BWL command applies to all channels; when off, the command is used to set the bandwidth individually for each channel. The response to the BANDWIDTH_LIMIT? Query indicates whether the bandwidth filters are on or off.

COMMAND SYNTAX

BandWidth_Limit <mode>

Or, alternatively, to choose the bandwidth limit of an individual channel or channels when Global_BWL is *off*:

BandWidth_Limit <channel>**,**<mode>[**,**<channel>**,**<mode>
[**,**<channel>**,**<mode>[**,**<channel>**,**<mode>]]]

<mode> : = {**OFF**, **ON**, **200MHZ**}

<channel> : = {**C1**, **C2**}

QUERY SYNTAX

BandWidth_Limit?

RESPONSE FORMAT

When Global_BWL is *on*, or if Global_BWL is *off* and all channels have the same bandwidth limit, the response is:

BandWidth_Limit <mode>

Or, alternatively, if at least two channels have their bandwidth limit filters set differently from one another, the response is:

BandWidth_Limit <channel>**,**<mode>[**,**<channel>**,**<mode>
[**,**<channel>**,**<mode>[**,**<channel>**,**<mode>]]]

EXAMPLE

The following turns on the bandwidth filter for all channels:

BWL ON

The following turns the bandwidth filter on for Channel 1 only (the first instruction turns *off* Global_BWL):

GBWL OFF

BWL C1,ON

RELATED COMMANDS

GLOBAL_BWL

MISCELLANEOUS *CAL?
Query

DESCRIPTION

The *CAL? query cause the LSA1000 to perform an internal self-calibration and generates a response that indicates whether or not the instrument completed the calibration without error. This internal calibration sequence is the same as that which occurs at power-up. At the end of the calibration, after the response has indicated how the calibration terminated, the instrument returns to the state it was in just prior to the calibration cycle.

QUERY SYNTAX

```
*CAL?
```

RESPONSE FORMAT

`*CAL` <diagnostics>

<diagnostics> : = **`0`** or other

`0` = Calibration successful

EXAMPLE

The following instruction forces a self-calibration:

```
*CAL?
```

Response message (if no failure): **`*CAL 0`**

RELATED COMMANDS

AUTO_CALIBRATE, CAL_STATUS

The *CAL? query performs an internal calibration and returns an integer denoting any errors.

Bit	Value	Description
0	1	C1 failure
1	2	C2 failure
2	4	C3 failure
3	8	C4 failure
4	16	TDC failure
5	32	Trigger failure
6	64	Other failure
7	128	reserved

MISCELLANEOUS CAL_MARGIN?, CMGN?

Query

DESCRIPTION

The CAL_MARGIN? query checks the margins on all the device-specific internal adjustments and responds with a list of all the names and margins. Margin values close to 0% or 100% are bad. Ideal margin values are around 50%.

QUERY SYNTAX

`Cal_MarGiN` [<tag>, <margin>[, <tag>, <margin>[,…]]]

<margin>: = <percentage in range 0–100>

<tag>:= tag string denoting what the margin is for.

EXAMPLE

The following instruction reads all of the margins:

The query and response

```
CMGN?
```

```
CMGN C1_A_DELAY,50.2 PCT,C1_A_GAIN_MATCH,50.2
PCT,C1_A_OFST_MATCH,50.2 PCT,C1_B_DELAY,50.2
PCT,C1_B_GAIN_MATCH,50.2 PCT,C1_B_OFST_MATCH,50.2
PCT,C1_GAIN,50 PCT,C1_OFFSET,50
PCT,C1_TRIG_THRESH1,50.2 PCT,C1_TRIG_THRESH2,50.2
PCT,C2_A_DELAY,50.2 PCT,C2_A_GAIN_MATCH,50.2
PCT,C2_A_OFST_MATCH,50.2 PCT,C2_B_DELAY,50.2
PCT,C2_B_GAIN_MATCH,50.2 PCT,C2_B_OFST_MATCH,50.2
PCT,C2_GAIN,50 PCT,C2_OFFSET,50
PCT,C2_TRIG_THRESH1,50.2 PCT,C2_TRIG_THRESH2,50.2
PCT,PS_+12_VCO,40.4 PCT,PS_+3.3,47.3 PCT,PS_+5VCO,50
PCT,PS__+5V_FE,44.9 PCT,P_+6V,62.8 PCT,PS_-2V,50.7
PCT,PS_-3.8VCO,48.9 PCT,PS_-4.5V_ADC,48 PCT,PS_-
5V_FEC,42.9 PCT,PS_-5V_FED,46.9 PCT,PS_-5V_MSH,38.8
PCT,PS_-6VC,47.8 PCT,PS_-6VD,43.7 PCT,PS_-VT,51.7
PCT,PS_12V_FAN,53.6 PCT,PS_2V5,44.4 PCT,PS_3V3,51.3
PCT,PS_VCC,37.8 PCT,PS_VEE,35.8 PCT
```

RELATED COMMANDS

AUTO_CALIBRATE, *CAL?

MISCELLANEOUS **CAL_STATUS?, CSTS?**

Query

DESCRIPTION

The CAL_STATUS? query checks the margins on all the internal controls and the status of the most recent calibration and returns an integer denoting any errors or warnings.

Bit	Value	Description
0–7	**0..255**	Same as for *CAL? query
8	**256**	Calibration recommended. This bit is set when automatic calibrations are disabled (using the AUTO_CALIBRATE command) and a temperature change or time period has elapsed that would ordinarily trigger an automatic calibration.
9	**512**	Margin violations detected. This bit is set when one or more of the internal controls used to maintain the calibration of the unit is currently within 10% of the end of its adjustment range. A margin violation may accompany a calibration failure (as reported in bits 0–7 or by the *CAL? command) but does not by itself necessarily indicate an internal hardware failure or that the unit is not able to perform to specifications. A margin violation not accompanied by a calibration failure indicates that the calibration passed conditionally. The unit should still function, but the warning indicates that some internal adjustments are near the end of their range. The criteria for a margin violation are intentionally more strict than the criteria for reporting a calibration failure. Thus, even properly functioning hardware may occasionally trigger a margin violation either due to an anomalous reading during a calibration or due to factors such as temperature transients (such as while the unit is warming up after power on). Frequent and persistent margin violations can be caused by extreme operating conditions (for example, extreme temperature), drift in the hardware that is exceeding the unit's ability to compensate internally, or some other hardware failure.

QUERY SYNTAX

`Cal_STatuS?`

RESPONSE FORMAT

`Cal_STatuS` <diagnostics>

<diagnostics>: = 0 to 1023

`0` = Calibration status OK.

EXAMPLE

The following instruction reads the calibration status:

CSTS?

Response message

CSTS 256

Indicates that a calibration is recommended

RELATED COMMANDS

AUTO_CALIBRATE, CAL?, CAL_MARGIN?

FUNCTION CLEAR_MEMORY, CLM

Command

DESCRIPTION

The CLEAR_MEMORY command clears the specified memory. Data previously stored in this memory are erased and memory space is returned to the free memory pool.

COMMAND SYNTAX

`CLear_Memory` < memory>

<memory> : = {**`M1`**, **`M2`**, **`M3`**, **`M4`**}

EXAMPLE

The following command clears the memory M2.

```
CLM M2
```

RELATED COMMANDS

STORE

FUNCTION	**CLEAR_SWEEPS, CLSW** **Command**
DESCRIPTION	The CLEAR_SWEEPS command restarts the cumulative processing functions: summed or continuous average, extrema, FFT power average, histogram, pulse parameter statistics, pass/fail counters, and persistence.
COMMAND SYNTAX	`CLear SWeeps`
EXAMPLE	The following example will restart the cumulative processing: `CLSW`
RELATED COMMANDS	DEFINE, INR

STATUS — *CLS Command

DESCRIPTION The *CLS command clears all the status data registers.

COMMAND SYNTAX **`*CLS`**

EXAMPLE The following command causes all the status data registers to be cleared:

`*CLS`

RELATED COMMANDS ALL_STATUS, CMR, DDR, *ESR, EXR, *STB, URR

STATUS CMR?

Query

DESCRIPTION

The CMR? query reads and clears the contents of the CoMmand error Register (CMR) — *see table next page* — which specifies the last syntax error type detected by the instrument.

QUERY SYNTAX

`CMR?`

RESPONSE FORMAT

`CMR` <value>

<value> : = 0 to 13

EXAMPLE

The following instruction reads the contents of the CMR register:

```
CMR?
```

Response message:

```
CMR 0
```

RELATED COMMANDS

ALL_STATUS?, *CLS

ADDITIONAL INFORMATION

Command Error Status Register Structure (CMR)	
Value	**Description**
1	Unrecognized command/query header
2	Illegal header path
3	Illegal number
4	Illegal number suffix
5	Unrecognized keyword
6	String error
7	GET embedded in another message
10	Arbitrary data block expected
11	Non-digit character in byte count field of arbitrary data block
12	EOI detected during definite length data block transfer
13	Extra bytes detected during definite length data block transfer

DISPLAY — COLOR, COLR
Command/Query

DESCRIPTION

The COLOR command is used to select the color of an individual display object such as text, trace, grid or cursor.

The response to the COLOR? query indicates the color assigned to each display object, whether or not it is currently displayed.

Note: This command is only effective if the color scheme (CSCH) is chosen from U1…U4.

This command is included for used with programs such as ScopeExplorer.

COMMAND SYNTAX

COLoR <object, color>[,...<object>,<color>]

<object> : = {**BACKGND**, **C1**, **C2**, **C3**, **C4**, **TA**, **TB**, **TC**, **TD**, **GRID**, **TEXT**, **CURSOR**, **NEUTRAL**, **WARNING**},

<color> : = { **WHITE**, **CYAN**, **YELLOW**, **GREEN**, **MAGENTA**, **BLUE**, **RED**, **LTGRAY**, **GRAY**, **SLGRAY**, **CHGRAY**, **DKCYAN**, **CREAM**, **SAND**, **AMBER**, **OLIVE**, **LTGEEN**, **JADE**, **LMGREEN**, **APGREEN**, **EMGREEN**, **GRGREEN**, **OCSPRAY**, **ICEBLUE**, **PASTBLUE**, **PALEBLUE**, **SKYBLUE**, **ROYLBLUE**, **DEEPBLUE**, **NAVY**, **PLUM**, **PURPLE**, **AMETHYST**, **FUCHSIA**, **RASPBRY**, **NEONPINK**, **PALEPINK**, **PINK**, **VERMIL**, **ORANGE**, **CERISE**, **KHAKI**, **BROWN**, **BLACK**}

QUERY SYNTAX

COLoR?

RESPONSE FORMAT

COLoR <object>,<color> [,...<object>,<color>]

EXAMPLE

The following instruction selects color scheme U1, and then red as the color of Channel 1:

```
CSCH U1
COLR C1,RED
```

RELATED COMMANDS

COLOR_SCHEME, PERSIST_COLOR

ADDITIONAL INFORMATION

Notation			
<color>	**Color**	**<color>**	**Color**
WHITE	White	**OCSPRAY**	Ocean Spray
CYAN	Cyan	**ICEBLUE**	Ice Blue
YELLOW	Yellow	**PASTBLUE**	Pastel Blue
GREEN	Green	**PALEBLUE**	Pale Blue
MAGENTA	Magenta	**SKYBLUE**	Sky Blue
BLUE	Blue	**ROYLBLUE**	Royal Blue
RED	Red	**DEEPBLUE**	Deep Blue
LTGRAY	Light Gray	**NAVY**	Navy
GRAY	Gray	**PLUM**	Plum
SLGRAY	Slate Gray	**PURPLE**	Purple
CHGRAY	Charcoal Gray	**AMETHYST**	Amethyst
DKCYAN	Dark Cyan	**FUCHSIA**	Fuchsia
CREAM	Cream	**RASPB**	Raspberry
SAND	Sand	**NEONPINK**	Neon Pink
AMBER	Amber	**PALEPINK**	Pale Pink
OLIVE	Olive	**PINK**	Pink
LTGREEN	Light Green	**VERMIL**	Vermilion
JADE	Jade	**ORANGE**	Orange
LMGREEN	Lime Green	**CERISE**	Cerise
APGREEN	Apple Green	**KHAKI**	Khaki
EMGREEN	Emerald Green	**BROWN**	Brown
GRGREEN	Grass Green	**BLACK**	Black
<object>	**Display Object**	**<object>**	**Display Object**
BACKGND	Background	**CURSOR**	cursors
C1..C4	Channel Traces	**WARNING**	Warning Messages
TA..TD	Function Traces	**NEUTRAL**	Neutral color
GRID	Grid lines	**OVERLAYS**	Menu background color (Full Screen)

DISPLAY COLOR_SCHEME, CSCH

Command/Query

DESCRIPTION

The COLOR_SCHEME command is used to select the color scheme for the display.

The response to the COLOR_SCHEME? query indicates the color scheme in use.

This command is included for used with programs such as ScopeExplorer.

COMMAND SYNTAX

`Color_SCHeme` <scheme>

<scheme > : = {**1**, **2**, **3**, **4**, **5**, **6**, **7**, **U1**, **U2**, **U3**, **U4**}

QUERY SYNTAX

`Color_SCHeme?`

RESPONSE FORMAT

`Color_SCHeme` <scheme>

EXAMPLE

The following instruction selects the user color scheme U2:

```
CSCH U2
```

RELATED COMMANDS

COLOR, PERSIST_COLOR

ACQUISITION COMBINE_CHANNELS, COMB

Command/Query

DESCRIPTION

The COMBINE_CHANNELS command controls the channel interleaving function of the acquisition system. The COMBINE_CHANNELS? query returns the interleaving function's current status.

COMMAND SYNTAX

`COMBine_channels` <state>

<state> : = {**1**, **2**}

QUERY SYNTAX

`COMBine_channels?`

RESPONSE FORMAT

`COMB <state>`

EXAMPLE

The following engages the interleaving function:

```
COMB 2
```

RELATED COMMANDS

COMBINE_SOURCE

ACQUISITION COMBINE_SOURCE, COMS

Command/Query

DESCRIPTION

The COMBINE_SOURCE command controls which input channel is used for interleaving.

The COMBINE_SOURCE? query returns the current setting.

COMMAND SYNTAX

`COMbine_Source` <channel>

<channel> : = {**`C1`**, **`C2`**}

QUERY SYNTAX

`COMbine_Source?`

RESPONSE FORMAT

`COMS <channel>`

EXAMPLE

The following sets the interleaved source to Channel 1:

```
COMS C1
```

RELATED COMMANDS

COMBINE_CHANNELS

COMMUNICATION COMM_FORMAT, CFMT

Command/Query

DESCRIPTION

The COMM_FORMAT command selects the format the LSA1000 uses to send waveform data. The available options allow the block format, the data type and the encoding mode to be modified from the default settings.

The COMM_FORMAT? query returns the currently selected waveform data format.

COMMAND SYNTAX

Comm_ForMaT <block_format>**,**<data_type>**,**<encoding>

<block_format> : = {**DEF9**, **IND0**, **OFF**}
<data_type> : = {**BYTE**, **WORD**}
<encoding> : = {**BIN**, **HEX**}

(ETHERNET uses both encoding forms)

Initial settings (i.e. after power-on) are:
DEF9, **WORD**, **BIN**

QUERY SYNTAX

Comm_ForMaT?

RESPONSE FORMAT

Comm_ForMaT <block_format>,<data_type>,<encoding>

EXAMPLE

The following code redefines the transmission format of waveform data. The data will be transmitted as a block of indefinite length. Data will be coded in binary and represented as 8-bit integers.

```
CFMT IND0,BYTE,BIN
```

ADDITIONAL INFORMATION **BLOCK FORMAT**

DEF9: Uses the IEEE 488.2 definite length arbitrary block response data format. The digit 9 indicates that the byte count consists of 9 digits. The data block directly follows the byte count field.

For example, a data block consisting of three data bytes would be sent as:

WF DAT1,#9000000003<DAB><DAB><DAB>

where <DAB> represents an eight-bit binary data byte.

IND0: Uses the IEEE 488.2 indefinite length arbitrary block response data format.

A <NL^END> (new line with EOI) signifies that block transmission has ended.

The same data bytes as above would be sent as:

`WF DAT1,#0`<DAB><DAB><DAB><NL^END>

OFF: Same as IND0. In addition, the data block type identifier and the leading #0 of the indefinite length block will be suppressed. The data presented above would be sent as:

`WF` <DAB><DAB><DAB><NL^END>

Note: The format OFF does not conform to the IEEE 488.2 standard and is only provided for special applications where the absolute minimum of data transfer may be important.

DATA TYPE

BYTE: Transmits the waveform data as eight-bit signed integers (one byte).

WORD: Transmits the waveform data as 16-bit signed integers (two bytes).

Note: The data type BYTE transmits only the high-order bits of the internal 16-bit representation. The precision contained in the low-order bits is lost.

ENCODING

BIN: Binary encoding

HEX: Hexadecimal encoding (bytes are converted to two hexadecimal ASCII digits (0, ...9, A, ...F)

RELATED COMMANDS WAVEFORM

COMMUNICATION COMM_HEADER, CHDR

Command/Query

DESCRIPTION

The COMM_HEADER command controls the way the LSA1000 formats responses to queries. The instrument provides three response formats: LONG format, in which responses start with the long form of the header word; SHORT format, where responses start with the short form of the header word; and OFF, for which headers are omitted from the response and suffix units in numbers are suppressed. Until the user requests otherwise, the SHORT response format is used.

This command does not affect the interpretation of messages sent to the LSA1000. Headers may be sent in their long or short form regardless of the COMM_HEADER setting.

Querying the vertical sensitivity of Channel 1 may result in one of the following responses:

COMM_HEADER	Response
LONG	C1:VOLT_DIV 200E-3 V
SHORT	C1:VDIV 200E-3 V
OFF	200E-3

COMMAND SYNTAX

Comm_HeaDeR <mode>

<mode> : = {**SHORT**, **LONG**, **OFF**}

Note: The default mode, i.e. the mode just after power-on, is SHORT.

QUERY SYNTAX

Comm_HeaDeR?

RESPONSE FORMAT

Comm_HeaDeR <mode>

EXAMPLE

The following code sets the response header format to SHORT:

```
CHDR SHORT
```

RELATED COMMANDS

COMM_HELP_LOG

COMMUNICATION COMM_HELP, CHLP

Command/Query

DESCRIPTION	The COMM_HELP command controls the level of operation of the diagnostics utility Remote Control Assistant, which assists in debugging remote control programs. Selected when using the instrument's front-panel via the "UTILITIES" and "SPECIAL MODES" menus, Remote Control Assistant can log all message transactions occurring between the external controller and the LSA1000. The log may be viewed at any time in the provided menu on the screen and has four levels to choose from:

OFF	Don't assist at all.
EO	Log detected Errors Only (default after power-on).
FD	Log the Full Dialog between the controller and the LSA1000.

COMMAND SYNTAX

`Comm_HeLP` <level>

<level> : = { **`OFF`**, **`EO`**, **`FD`** }

The default level (i.e. the level just after power-on) is EO.

QUERY SYNTAX

`Comm_HeLP?`

RESPONSE FORMAT

`Comm_HeLP` <level>

EXAMPLE (GPIB)

After sending this command, all the following commands and responses will be logged:

```
CHLP FD
```

RELATED COMMANDS

COMM_HELP_LOG

COMMUNICATION COMM_HELP_LOG?, CHL?

Query

DESCRIPTION

The COMM_HELP_LOG query returns the current contents of the log generated by the Remote Control Assistant (*see CHLP description*). If the optional parameter CLR is specified, the log will be cleared after the transmission. Otherwise, it will be kept.

QUERY SYNTAX

`Comm_HeLP_Log?` [**`CLR`**]

RESPONSE FORMAT

`Comm_Help_Log` <string containing the logged text>

EXAMPLE (GPIB)

The following code reads the remote control log and prints it:

```
CHL?
```

RELATED COMMANDS

COMM_HELP

COMMUNICATION COMM_NET, CONET
Command/Query

DESCRIPTION

The COMM_NET command specifies the network address of the instrument. The COMM_NET? query returns the current network address.

COMMAND SYNTAX

COmm_NET <subaddress>

<subaddress> : = {**IP**, **“X.X.X.X.”**, **MASK**, **”X.X.X.X.”**, **GATEWAY**, **“X.X.X.X”**}

QUERY SYNTAX

COmm_NET?

RESPONSE FORMAT

COmm_NET <subaddress>

EXAMPLE

________________**WARNING**______________

This command acts immediately. The software used to send the command will need to be initialized with the new address before continuing.

The query and response:

```
CONET?
IP,”172.25.1.2”,MASK,”255.255.0.0”,GATEWAY,”172.25
.0.1”
```

This command changes the IP:

```
COMM_NET IP,”172.28.11.77”
```

COMMUNICATION — COMM_ORDER, CORD

Command/Query

DESCRIPTION

The COMM_ORDER command controls the byte order of waveform data transfers. Waveform data may be sent with the most significant byte (MSB) or the least significant byte (LSB) in the first position. The default mode is the MSB first.

COMM_ORDER applies equally to the waveform's descriptor and time blocks. In the descriptor some values are 16 bits long ("word"), 32 bits long ("long "or "float"), or 64 bits long ("double"). In the time block all values are floating values, i.e. 32 bits long. When "COMM_ORDER HI" is selected, the most significant byte is sent first. When "COMM_ORDER LO" is specified, the least significant byte is sent first.

The COMM_ORDER? query returns the byte transmission order currently in use.

COMMAND SYNTAX

`Comm_ORDer` <mode>

<mode> : = {**`HI`**, **`LO`**}

Note: The initial mode, i.e. the mode after power-on, is HI.

QUERY SYNTAX

`Comm_ORDer?`

RESPONSE FORMAT

`Comm_ORDer` <mode>

EXAMPLE

The order of transmission of waveform data depends on the data type. The following table illustrates the different possibilities.

Type	CORD HI	CORD LO
Word	<MSB><LSB>	<LSB><MSB>
Long/Float	<MSB><byte2><byte3><LSB>	<LSB><byte3><byte2><MSB>
Double	<MSB><byte2>...<byte7><LSB>	<LSB><byte7>...<byte2><MSB>

RELATED COMMANDS

WAVEFORM

CURSOR — CURSOR_MEASURE, CRMS

Command/Query

DESCRIPTION

The CURSOR_MEASURE command specifies the type of cursor or parameter measurement to be displayed, and is the main command for displaying parameters and pass/fail.

The CURSOR_MEASURE? query indicates which cursors or parameter measurements are currently displayed.

Notation	
ABS	absolute reading of relative cursors
CUST	custom parameters
FAIL	pass/fail: fail
HABS	horizontal absolute cursors
HPAR	standard time parameters
HREL	horizontal relative cursors
OFF	cursors and parameters off
PARAM	synonym for VPAR
PASS	pass/fail: pass
SHOW	custom parameters (old form)
STAT	parameter statistics
VABS	vertical absolute cursors
VPAR	standard voltage parameters
VREL	vertical relative cursors

Note: The PARAM mode is turned OFF when the XY mode is ON.

COMMAND SYNTAX

CuRsor_MeaSure <mode>[,<submode>]

<mode> : = {**CUST**, **FAIL**, **HABS**, **HPAR**, **HREL**, **OFF**, **PARAM**, **PASS**, **SHOW**, **VABS**, **VPAR**, **VREL**}

<submode> : = {**STAT**, **ABS**}

Note 1: The keyword STAT is optional with modes CUST, HPAR, and VPAR. If present, STAT turns parameter statistics on. Absence of STAT turns parameter statistics off.

Note 2: The keyword ABS is optional with mode HREL. If it is present, ABS chooses absolute amplitude reading of relative cursors. Absence of ABS selects relative amplitude reading of relative cursors.

QUERY SYNTAX **CuRsor_MeaSure?**

RESPONSE FORMAT **CuRsor_MeaSure** <mode>

EXAMPLE The following command switches on the vertical relative cursors:

CRMS VREL

The following command determines which cursor is currently turned on:

CRMS?

Example of response message:

CRMS OFF

RELATED COMMANDS CURSOR_SET, PARAMETER_STATISTICS, PARAMETER_VALUE, PASS_FAIL_CLEAR, PASS_FAIL_CONDITION, PASS_FAIL_DELETE, PASS_FAIL_MASK,

ADDITIONAL INFORMATION To turn off the cursors, parameter measurements or Pass/Fail tests, use:

```
CURSOR_MEASURE OFF
```

To turn on a cursor display, use one of the following four forms:

```
CURSOR_MEASURE HABS
CURSOR_MEASURE HREL
CURSOR_MEASURE VABS
CURSOR_MEASURE VREL
```

To turn on parameter measurements without statistics, use one of the following three forms:

```
        CURSOR_MEASURE CUST
        CURSOR_MEASURE HPAR
        CURSOR_MEASURE VPAR
```

To turn on parameter statistics, add the keyword **`STAT`** to the above three forms.

To turn on Pass or Fail tests on parameter or mask tests, use:

```
        CURSOR_MEASURE PASS
        CURSOR_MEASURE FAIL
```

Use the command:

```
        PASS_FAIL_CONDITION
```

to select parameters in the Custom mode, and to modify the test conditions in the Pass/Fail mode.

CURSOR — CURSOR_SET, CRST

Command/Query

DESCRIPTION

The CURSOR_SET command allows the user to position any one of the eight independent 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 on the screen.

When setting a cursor position, a trace must be specified, relative to which the cursor will be positioned.

The CURSOR_SET? query indicates the current position of the cursor(s). The values returned depend on the grid type selected.

Note: To change only the trace without repositioning the cursors, the CURSOR_SET command may be given with no argument (for example,. TB:CRST).

Notation			
HABS	horizontal absolute	**PREF**	parameter reference
HDIF	horizontal difference	**VABS**	vertical absolute
HREF	horizontal reference	**VDIF**	vertical difference
PDIF	parameter difference	**VREF**	vertical reference

COMMAND SYNTAX

<trace>:**CuRsor_SeT** <cursor>,<position>[,<cursor>,<position>,<cursor>,<position>]

<trace> : = {**TA**, **TB**, **TC**, **TD**, **C1**, **C2**}

<cursor> : = {**HABS**, **VABS**, **HREF**, **HDIF**, **VREF**, **VDIF**, **PREF**, **PDIF**}

<position> : = 0 to 10 DIV (horizontal)

<position> : = –29.5 to 29.5 DIV (vertical)

Note 1: The suffix DIV is optional.

Note 2: Parameters are grouped in pairs. The first parameter specifies the cursor to be modified and the second one indicates its new value. Parameters may be grouped in any order and may be restricted to those items to be changed.

QUERY SYNTAX

<trace> : **CuRsor_SeT?** [<cursor>,...<cursor>]

<cursor> : = {**HABS**, **VABS**, **HREF**, **HDIF**, **VREF**, **VDIF**, **PREF**, **PDIF**, **ALL**}

RESPONSE FORMAT

<trace> : **CuRsor_SeT** <cursor>**,**<position>[**,**<cursor>**,**<position>**,**... <cursor>**,**<position>]

If <cursor> is not specified, ALL will be assumed. If the position of a cursor cannot be determined in a particular situation, its position will be indicated as UNDEF.

EXAMPLE

The following command positions the VREF and VDIF cursors at +3 DIV and –7 DIV respectively, using Trace A as a reference:

TA:CRST VREF,3DIV,VDIF,- 7DIV

RELATED COMMANDS

CURSOR_MEASURE, CURSOR VALUE, PARAMETER_VALUE, PER_CURSOR_SET, XY_CURSOR_SET

CURSOR — CURSOR_VALUE?, CRVA?

Query

DESCRIPTION

The CURSOR_VALUE? query returns the values measured by the specified cursors for a given trace. (The PARAMETER_VALUE? query is used to obtain measured waveform parameter values.)

Notation			
HABS	horizontal absolute	**VABS**	vertical absolute
HREL	horizontal relative	**VREL**	vertical relative

QUERY SYNTAX

<trace> : **CuRsor_VAlue?** [<mode>**,**...<mode>]

<trace> : = {**TA**, **TB**, **TC**, **TD**, **C1**, **C2**}
<mode> : = {**HABS**, **HREL**, **VABS**, **VREL**, **ALL**}

RESPONSE FORMAT

<trace> : **CuRsor_VAlue HABS,**<abs_hori>**,**<abs_vert>
<trace> : **CuRsor_VAlue HREL,**<delta_hori>**,**<delta_vert>**,**
<absvert_ref>**,**<absvert_dif>
<trace> : **CuRsor_VAlue VABS,**<abs_vert>
<trace> : **CuRsor_VAlue VREL,**<delta_vert>

For horizontal cursors, both horizontal as well as vertical values are given. For vertical cursors only vertical values are given.

Note: If <mode> is not specified or equals ALL, all the measured cursor values for the specified trace are returned. If the value of a cursor cannot be determined in the current environment, the value UNDEF will be returned.

EXAMPLE

The following query reads the measured absolute horizontal value of the cross-hair cursor (HABS) on Channel 2:

C2:CRVA? HABS

Response message:
C2:CRVA HABS,34.2E-6 S, 244 E-3 V

RELATED COMMANDS

CURSOR_SET, PARAMETER_VALUE, PER_CURSOR_VALUE, XY_CURSOR_VALUE

DISPLAY DATA_POINTS, DPNT

Command/Query

DESCRIPTION

The DATA_POINTS command is used to control whether the waveform sample points are shown as single display pixels or are made bold.

The response to the DATA_POINTS? query indicates whether the waveform sample points are being displayed as single pixels or in bold face.

This command is included for used with programs such as ScopeExplorer.

COMMAND SYNTAX

`Data_PoiNTs` <state>
<state> : = {**`NORMAL, BOLD`**}

QUERY SYNTAX

`Data_PoiNTs?`

RESPONSE FORMAT

`DataPoiNTs` <state>

EXAMPLE

The following instruction highlights the waveform sample points:

```
DPNT BOLD
```

MISCELLANEOUS — DATE

Command/Query

DESCRIPTION

The DATE command changes the date/time of the LSA1000's internal real-time clock.

The DATE? query returns the current date/time setting.

COMMAND SYNTAX

DATE <day>**,**<month>**,**<year>**,**<hour>**,**<minute>**,**<second>

<day> : = 1 to 31

<month> : = {**JAN**, **FEB**, **MAR**, **APR**, **MAY**, **JUN**, **JUL**, **AUG**, **SEP**, **OCT**, **NOV**, **DEC**}

<year> : = 1990 to 2089

<hour> : = 0 to 23

<minute> : = 0 to 59

<second> : = 0 to 59

Note: It is not always necessary to specify all the DATE parameters. Only those parameters up to and including the parameter to be changed need be specified in order to change the "year" setting, specify day, month and year together with the required settings. The time settings will remain unchanged. To change the "second" setting, all the DATE parameters must be specified with the required settings.

QUERY SYNTAX

DATE?

RESPONSE FORMAT

DATE <day>,<month>,<year>,<hour>,<minute>,<second>

EXAMPLE

This instruction will change the date to January 1, 1997 and the time to 1:21:16 p.m. (13:21:16 in 24-hour notation):

```
DATE 1,JAN,1997,13,21,16
```

STATUS DDR?

Query

DESCRIPTION

The DDR? query reads and clears the contents of the Device Dependent or device specific error Register (DDR). In the case of a hardware failure, the DDR register specifies the origin of the failure. The following table gives details.

Bit	Bit Value	Description	
15...14		0	Reserved
13	8192	1	Timebase hardware failure detected
12	4096	1	Trigger hardware failure detected
11	2048	0	Reserved
10	1024	0	Reserved
9	512	1	Channel 2 hardware failure detected
8	256	1	Channel 1 hardware failure detected
7	128	1	External input overload condition detected
6...4		0	Reserved
3	8	0	Reserved
2	4	0	Reserved
1	2	1	Channel 2 overload condition detected
0	1	1	Channel 1 overload condition detected

QUERY SYNTAX

```
DDR?
```

RESPONSE FORMAT

`DDR` <value>

<value> : = 0 to 65535

EXAMPLE

The following instruction reads the contents of the DDR register:

```
DDR
```

Response message:

```
DDR 0
```

RELATED COMMANDS

ALL_STATUS, *CLS

FUNCTION DEFINE, DEF

Command/Query

DESCRIPTION

The DEFINE command specifies the mathematical expression to be evaluated by a function. This command is used to control all functions in the standard instruments and WP0X processing packages.

COMMAND SYNTAX

<function> **: DEFine EQN,'**<equation>**'** [**,**<param_name>**,**<value>**,**...]

Note 1: Parameters are grouped in pairs. The first in the pair names the variable to be modified, <param_name>, while the second one gives the new value to be assigned. Pairs can be given in any order and restricted to the variables to be changed.

Note 2: Space (blank) characters inside equations are optional.

Note 3: The LSA1000 only calculates functions that have a consumer. That is, either the function trace must be on (see the TRACE command, page 131) or the function must be used by another function or a parameter.

QUERY SYNTAX

<function> **: DEFine?**

RESPONSE FORMAT

<function> **: DEFine EQN,'**<equation>**'**[**,MAXPTS,**<max_points>] [**,SWEEPS,**<max_sweeps>][**,WEIGHT,**<weight>][**,BITS,**<bits>]

<param_name>	<value>	Description
EQN	'<equation>'	Function equation as defined below
MAXPTS	<max_points>	Maximum number of points to compute
SWEEPS	<max_sweeps>	Maximum number of sweeps
Parameters To Support Additional Functions in WP01		
WEIGHT	<weight>	Continuous Average weight
BITS	<bits>	Number of ERES bits
Parameters To Support Additional Functions in WP02		
WINDOW	<window_type>	FFT window function

Parameters To Support Additional Functions in WP03 or DDM		
MAXBINS	<bins>	Number of bins in histogram
MAX_EVENTS	<max_values>	Maximum number of values in histogram
CENTER	<center>	Horizontal center position for histogram display.
WIDTH	<width>	Width of histogram display
VERT	<vert_scale>	Vertical scaling type

Parameters To Support Additional Functions in PRML		
LENGTH	<length>	Number of points to use from first waveform
START	<start>	Starting point in second waveform

Function Equations And Names Available On All Models	
<source>	Identity
+<source>	Identity
-<source>	Negation
<source1> + <source2>	Addition
<source1> - <source2>	Subtraction
<source1><source2>	Multiplication
<source1>/<source2>	Ratio
AVGS(<source>)	Average Summed
SINX(<source>)	Sin(x)/x interpolator
ZOOMONLY (<extended_source>)	Zoom only (No Math)

Extended Functions Available On Instruments With WP01 Processing Firmware	
ABS(<source>)	Absolute Value
AVGC(<source>)	Continuous Average
DERI(<source>)	Derivative
ERES(<source>)	Enhanced Resolution
EXP(<source>)	Exponential (power of e)
EXP10(<source>)	Exponential (power of 10)
EXTR(<source>)	Extrema (Roof and Floor)
FLOOR(EXTR(<source>))	Floor (Extrema source only)
INTG(<source>[{+,-} <addend>])	Integral

LN(<source>)	Logarithm base e	
LOG10(<source>)	Logarithm base 10	
RESC([{+,-}][<multiplier>*]<source>[{+,-}<addend>])	Rescale	
ROOF(EXTR(<source>))	Roof (Extrema source only)	
1/<source>	Reciprocal	
SQR(<source>)	Square	
SQRT(<source>)	Square Root	
FFT Functions Available on Instruments with WP02 Processing Firmware *Note: The source waveform must be a time-domain signal, single segment.*		
FFT(<source>)	Fast Fourier Transform (complex result)	
REAL(FFT(<source>))	Real part of complex result	
IMAG(FFT(<source>))	Imaginary part of complex result	
MAG(FFT(<source>))	Magnitude of complex result	
PHASE(FFT(<source>))	Phase angle (degrees) of complex result	
PS(FFT(<source>))	Power spectrum	
PSD(FFT(<source>))	Power density	
RESC([{+,-}][<multiplier>]<source>[{+,-}<addend>])	Rescale	
Power Average Functions Available on Instruments with WP02 Processing Firmware *Note: The source waveform must be another function defined as a Fourier transform.*		
MAG(AVGP(<function>))	PS(AVGP(<function>))	PSD(AVGP(<function>))
Function Equations and Names Available on Instruments with WP03 or DDM Firmware		
HIST(<custom_line>)	Histogram of parameter on custom line	
Function Equations and Names Available on Instruments with PRML Firmware		
CORR(<source1>,<source2>)	Cross Correlation	

Source values

Note: The numbers in CUST1, CUST2, CUST3, CUST4, and CUST5 refer to the line numbers of the selected custom parameters.

<sourceN> : = {**TA**, **TB**, **TC**, **TD**, **M1**, **M2**, **M3**, **M4**, **C1**, **C2**}

<function> : = {**TA**, **TB**, **TC**, **TD**}

<custom_line> : = {**CUST1, CUST2, CUST3, CUST4, CUST5**}

<extended_source> : = {**C1, C2, TA, TB, TC, TD, M1, M2, M3, M4**}

Values to define number of points/sweeps

<max_points> : = 50 to 10 000 000
<max sweeps> : = 1 to 1000 (For standard instruments)
<max_sweeps> : = 1 to 1 000 000 (For WP01 only)
<max_sweeps> : = 1 to 50 000 (WP02 Power Spectrum only)

Values for Rescale Function

<addend> : = 0.0 to 1e15
<multiplier> : = 0.0 to 1e15

Values for Summation Average and ERES

<weight> : = {**1**, **3**, **7**, **15**, **31**, **63**, **127**, **255**, **511**, **1023**}
<bits> : = {**0.5**, **1.0**, **1.5**, **2.0**, **2.5**, **3.0**}

Values for FFT window function

<window_type> : = {**BLHA**, **FLTP**, **HAMM**, **HANN**, **RECT**}

FFT Window Function Notation	
LHA	Blackman–Harris window
FLTP	Flat Top window
HABMM	Hamming window
HANN	von Hann window
RECT	Rectangular window

Values for WP03 histogramming

<max bins> : = {**20**, **50**, **100**, **200**, **500**, **1000**, **2000**}
<max_events> : = 20 to 2e9 (in a 1–2–5 sequence)
<center> : = –1e15 to 1e15
<width> : = 1e–30 to 1e30 (in a 1–2–5 sequence)
<vert_scale> : = {**LIN**, **LOG**, **CONSTMAX**}

Histogram Notation	
LIN	Use linear vertical scaling for histogram display
LOG	Use log vertical scaling for histogram display
CONSTMAX	Use constant maximum linear scaling for histogram display

Values for PRML correlation
<length> : = 0 to 10 divisions
<start> : = 0 to 10 divisions

☝ **AVAILABILITY**

SWEEPS is the maximum number of sweeps (Average and Extrema only).

Note: The pair SWEEPS,<max_sweeps> applies only to the summed averaging (AVGS).

EXAMPLE

The following instruction defines Trace A to compute the summed average of Channel 1 using 5000 points over 200 sweeps:

```
TA:DEF EQN,'AVGS(C1)',MAXPTS,5000,SWEEPS,200
```

WP01 EXAMPLE

The following instruction defines Trace A to compute the product of Channel 1 and Channel 2, using a maximum of 10 000 input points:

```
TA:DEF EQN,'C1*C2',MAXPTS,10000
```

WP02 FFT EXAMPLE

The following instruction defines Trace A to compute the Power Spectrum of the FFT of Channel 1. A maximum of 1000 points will be used for the input. The window function is Rectangular.

```
TA:DEF EQN,'PS(FFT(C1))',MAXPTS,1000,WINDOW,RECT
```

WP02 PS EXAMPLE

The following instruction defines Trace B to compute the Power Spectrum of the Power Average of the FFT being computed by Trace A, over a maximum of 244 sweeps.

```
TB:DEF EQN,‘PS(AVGP(TA))’,SWEEPS,244
```

WP03 EXAMPLE

The following command defines Trace C to construct the histogram of the all rise time measurements made on source Channel 1. The rise time measurement is defined on custom line 2. The histogram has a linear vertical scaling and the rise time parameter values are binned into 100 bins.

```
PACU 2,RISE,C1
TC:DEF EQN,‘HIST(CUST2)’,VERT,LIN,MAXBINS,100
```

RELATED COMMANDS

FIND_CTR_RANGE, FUNCTION_RESET, INR?, PARAMETER_CUSTOM, PARAMETER_VALUE?, PASS_FAIL_CONDITION, TRACE

MASS STORAGE DELETE_FILE, DELF

Command

DESCRIPTION

The DELETE_FILE command deletes files from the currently selected directory on mass storage.

COMMAND SYNTAX

`DELete_File DISK,`<device>**`,FILE,'`**<filename>**`'`**

<device> : = {**CARD**☝, **FLPY**☝, **HDD**☝}

<filename> : = An alphanumeric string of up to eight characters, followed by a dot and an extension of up to three characters.

☝ **AVAILABILITY**

<device> : CARD available only when MC01 option is fitted.

<device> : FLPY available only when FD01 option is fitted.

<device> : HDD available only when HD01 option is fitted.

EXAMPLE

The following command deletes a front-panel setup from the memory card:

```
DELF DISK,CARD,FILE,'P001.PNL'
```

RELATED COMMANDS

DIRECTORY, FORMAT_CARD, FORMAT_FLOPPY, FORMAT_HDD

MASS STORAGE — DIRECTORY, DIR

Command/Query

DESCRIPTION

The DIRECTORY command is used to manage the creation and deletion of file directories on mass storage devices. It also allows selection of the current working directory and listing of files in the directory.

The query response consists of a double-quoted string containing a DOS-like listing of the directory. If no mass storage device is present, or if it is not formatted, the string will be empty.

COMMAND SYNTAX

DIRectory DISK,<device>**,ACTION,**<action>**,'**<directory>**'**

QUERY SYNTAX

DIRectory? DISK,<device> [**,'**<directory>**'**]

<device> : = {**CARD**☝, **FLPY**☝, **HDD**☝}
<action> : = {**CREATE**, **DELETE**, **SWITCH**}
<directory> : = A legal DOS path or filename. (This can include the '\' character to define the root directory.)

Note: the query DIRectory_list? is also accepted for backward compatibility but may not be supported in the future.

RESPONSE FORMAT

DIRectory DISK,<device> **"**<directory>**"**

<directory> : = A variable length string detailing the file content of the memory card, floppy disk or hard disk.

☝ **AVAILABILITY**

<device> : CARD available only when MC01 option is fitted.
<device> : FLPY available only when FD01 option is fitted.
<device> : HDD available only when HD01 option is fitted.

EXAMPLE

The following asks for a listing of the directory of the memory card:

DIR? DISK,CARD

Response message:

```
DIR "
Directory         LECROY      1 DIR of 04-MAY-1998
  10:46:20 on Memory Card
SC1000    2859    19-DEC-1994 16:33:06
SC1001    2859    19-DEC-1994 16:34:32
TEST5      002    20359       12-MAY-1998
    13:34:12
3 File(s) 1948672 bytes free
"
```

DISPLAY — DISPLAY, DISP

Command/Query

DESCRIPTION

The DISPLAY command controls the display screen of programs such as ScopeExplorer When the user is remotely controlling the instrument and does not need to use the display map feature, it can be useful to switch off the display map update via the DISPLAY OFF command. This improves instrument response time, since the waveform graphic generation procedure is suppressed.

The response to the DISPLAY? query indicates the display state of the instrument.

Note: When the display has been set to OFF, the real-time clock and the message field update. However, the waveforms and associated texts remain unchanged.

COMMAND SYNTAX

`DISPlay` <state>

<state> : = {**`ON, OFF`**}

QUERY SYNTAX

`DISPlay?`

RESPONSE FORMAT

`DISPlay` <state>

EXAMPLE

The following instruction turns off the display generation.

```
DISP OFF
```

DISPLAY DOT_JOIN, DTJN

Command/Query

DESCRIPTION

The DOT_JOIN command controls the interpolation lines between data points. This command is included for use with programs such as ScopeExplorer.

COMMAND SYNTAX

`DoT_JoiN` <state>

<state> : = {**`ON, OFF`**}

QUERY SYNTAX

`DoT_JoiN?`

RESPONSE FORMAT

`DoT_JoiN` <state>

EXAMPLE

The following instruction turns off the interpolation lines:

```
DTJN OFF
```

DISPLAY DUAL_ZOOM, DZOM

Command/Query

DESCRIPTION

By setting DUAL_ZOOM ON, the horizontal magnification and positioning controls are applied to all expanded traces simultaneously. This command is useful if the contents of all expanded traces are to be examined at the same time.

The DUAL_ZOOM? query indicates whether multiple zoom is enabled or not.

Note: This command has the same effect as MULTI_ZOOM.

This command is included for used with programs such as ScopeExplorer.

COMMAND SYNTAX

`Dual_ZOoM` <mode>

<mode> : = {**`ON`**, **`OFF`**}

QUERY SYNTAX

`Dual_ZOoM?`

RESPONSE FORMAT

`Dual_ZOoM` <mode>

EXAMPLE

The following instruction turns dual zoom on:

`DZOM ON`

RELATED COMMANDS

HOR_MAGNIFY, HOR_POSITION, MULTI_ZOOM

STATUS ***ESE**

Command/Query

DESCRIPTION

The *ESE command sets the Standard Event Status Enable register (ESE). This command allows one or more events in the ESR register to be reflected in the ESB summary message bit (bit 5) of the STB register. *For an overview of the ESB defined events refer to the ESR table on page 58*.

The *ESE? query reads the contents of the ESE register.

COMMAND SYNTAX

***ESE** <value>

<value> : = 0 to 255

QUERY SYNTAX

***ESE?**

RESPONSE FORMAT

***ESE** <value>

EXAMPLE

The following instruction allows the ESB bit to be set if a user request (URQ bit 6, i.e. decimal 64) and/or a device dependent error (DDE bit 3, i.e. decimal 8) occurs. Summing these values yields the ESE register mask 64+8=72.

***ESE 72**

RELATED COMMANDS

*ESR

STATUS — *ESR? Query

DESCRIPTION

The *ESR? query reads and clears the contents of the Event Status Register (ESR). The response represents the sum of the binary values of the register bits 0 to 7. The table below gives an overview of the ESR register structure.

QUERY SYNTAX

`*ESR?`

RESPONSE FORMAT

`*ESR` <value>

<value> : = 0 to 255

EXAMPLE

The following instruction reads and clears the contents of the ESR register:

`*ESR`

Response message:

`*ESR 0`

RELATED COMMANDS

ALL_STATUS, *CLS, *ESE

ADDITIONAL INFORMATION

Standard Event Status Register (ES)					
Bit	**Bit Value**	**Bit Name**	**Description**		**Note**
15...8			0	reserved by IEEE 488.2	
7	128	PON	1	Power off-to-ON transition has occurred	(1)
6	64	URQ	1	User ReQuest has been issued	(2)
5	32	CME	1	CoMmand parser Error has been detected	(3)
4	16	EXE	1	EXecution Error detected	(4)
3	8	DDE	1	Device specific Error occurred	(5)
2	4	QYE	1	QuerY Error occurred	(6)
1	2	RQC	0	Instrument never requests bus control	(7)
0	1	OPC	0	OPeration Complete bit not used	(8)

Notes

(1) The Power On (PON) bit is always turned on (1) when the unit is powered up.

(2) The User Request (URQ) bit is set true (1) when a soft key is pressed. An associated register URR identifies which key was selected. For further details refer to the URR? query.

(3) The CoMmand parser Error bit (CME) is set true (1) whenever a command syntax error is detected. The CME bit has an associated CoMmand parser Register (CMR) which specifies the error code. Refer to the query CMR? for further details.

(4) The EXecution Error bit (EXE) is set true (1) when a command cannot be executed due to some device condition (e.g. LSA1000 in local state) or a semantic error. The EXE bit has an associated Execution Error Register (EXR) which specifies the error code. Refer to query EXR? for further details.

*(5) The Device specific Error (DDE) is set true (1) whenever a hardware failure has occurred at power-up, or execution time, such as a channel overload condition, a trigger or a timebase circuit defect. The origin of the failure may be localized via the DDR? or the self test *TST? query.*

(6) The Query Error bit (QYE) is set true (1) whenever (a) an attempt is made to read data from the Output Queue when no output is either present or pending, (b) data in the Output Queue has been lost, (c) both output and input buffers are full (deadlock state), (d) an attempt is made by the controller to read before having sent an <END>, (e) a command is received before the response to the previous query was read (output buffer flushed).

(7) The ReQuest Control bit (RQC) is always false (0), as the LSA1000 has no GPIB controlling capability.

*(8) The OPeration Complete bit (OPC) is set true (1) whenever *OPC has been received, since commands and queries are strictly executed in sequential order. The LSA1000 starts processing a command only when the previous command has been entirely executed.*

STATUS EXR?
Query

DESCRIPTION

The EXR? query reads and clears the contents of the EXecution error Register (EXR). The EXR register specifies the type of the last error detected during execution. *Refer to the table next page for further details.*

QUERY SYNTAX

EXR?

RESPONSE FORMAT

EXR <value>

<value> : = 21 to 64

EXAMPLE

The following instruction reads the contents of the EXR register:

EXR

Response message (if no fault):

EXR 0

RELATED COMMANDS

ALL_STATUS, *CLS

ADDITIONAL INFORMATION

Execution Error Status Register Structure (EXR)	
Value	**Description**
21	Permission error. The command cannot be executed in local mode.
22	Environment error. The instrument is not configured to correctly process a command.
23	Option error. The command applies to an option which has not been installed.
24	Unresolved parsing error.
25	Parameter error. Too many parameters specified.
26	Non-implemented command.
30	Hex data error. A non-hexadecimal character has been detected in a hex data block.
31	Waveform error. The amount of data received does not correspond to descriptor indicators.
32	Waveform descriptor error. An invalid waveform descriptor has been detected.
33	Waveform text error. A corrupted waveform user text has been detected.
34	Waveform time error. Invalid RIS or TRIG time data has been detected.
35	Waveform data error. Invalid waveform data have been detected.
36	Panel setup error. An invalid panel setup data block has been detected.
50	No mass storage present when user attempted to access it. *
51	Mass storage not formatted when user attempted to access it. *
53	Mass storage was write protected when user attempted to create, or a file, to delete a file, or to format the device. *
54	Bad mass storage detected during formatting. *
55	Mass storage root directory full. Cannot add directory. *
56	Mass storage full when user attempted to write to it. *
57	Mass storage file sequence numbers exhausted (999 reached). *
58	Mass storage file not found. *
59	Requested directory not found. *
61	Mass storage filename not DOS compatible, or illegal filename. *
62	Cannot write on mass storage because filename already exists. *

* *For LSA1000s fitted with floppy disk (FD01), memory card (MCO1) or hard disk (HD01) options.*

MASS STORAGE — FILENAME, FLNM☝

Command/Query

DESCRIPTION

The FILENAME command is used to change the default filename given to any traces, setups and hard copies when they are being stored to a mass storage device.

COMMAND SYNTAX

`FiLeNaMe TYPE,`<type>**`,FILE,'`**<filename>**`'`**

<type> : = {**`C1`**, **`C2`**, **`TA`**, **`TB`**, **`TC`**, **`TD`**, **`SETUP`**, **`HCOPY`** }

<filename> : = For C1 to TD, an alphanumeric string of up to eight characters forming a legal DOS filename. Up to five characters for SETUP and HCOPY.

Note: No extension can be specified, as this is automatically assigned by the LSA1000.

QUERY SYNTAX

`FiLeNaMe? TYPE,`<type>

<type> : = {**`ALL`**, **`C1`**, **`C2`**, **`TA`**, **`TB`**, **`TC`**, **`TD`**, **`SETUP`**, **`HCOPY`**}

RESPONSE FORMAT

`FiLeNaMe`
TYPE,<type>**,**FILE**,** **"**<filename>**"**[**,**TYPE**,**<type>**,**FILE**,** **"**<filename>**"**...**]**

☝ **AVAILABILITY**

Only available on LSA1000s fitted with the MC01, FD01 or HD01 options.

EXAMPLE

The following command designates channel 1 waveform files to be "TESTPNT6.xxx" where xxx is a numeric extension assigned by the LSA1000:

```
FLNM TYPE,C1, FILE, 'TESTPNT6'
```

RELATED COMMANDS

DIRECTORY, FORMAT_CARD, FORMAT_FLOPPY, FORMAT_HDD, DELETE_FILE

FUNCTION — FIND_CTR_RANGE, FCR
Command

DESCRIPTION

The FIND_CTR_RANGE command automatically sets the center and width of a histogram to best display the accumulated events.

COMMAND SYNTAX

<function> : **`Find_Ctr_Range`**

<function> : = {**`TA`**,**`TB`**,**`TC`**,**`TD`**}

AVAILABILITY

Command only available on LSA1000s fitted with the WP03 or DDM options.

EXAMPLE

Assuming that Trace A (TA) has been defined as a histogram of one of the custom parameters, the following example will determine the best center and width and then rescale the histogram:

```
TA:FCR
```

RELATED COMMANDS

DEFINE, PACU

MASS STORAGE — FORMAT_CARD, FCRD☝
Command/Query

DESCRIPTION

The FORMAT_CARD command formats the memory card according to the PCMIA/JEIDA standard with a DOS partition.

The FORMAT_CARD? query returns the status of the card.

COMMAND SYNTAX

Format_CaRD

QUERY SYNTAX

Format_CaRD?

RESPONSE FORMAT

Format_CaRD <card_status>[**,**<read/write>**,**<free_space>**,** <card_size>**,**<battery_status>]

<card_status> : = {**NONE**, **BAD**, **BLANK**, **DIR_MISSING**, **OK}**

<read/write> : = {**WP**, **RW**}

<free_space> : = A decimal number giving the number of bytes still available on the card

<card_size> : = A decimal number giving the total number of bytes on the card.

<battery_status> : = {**BAT_OK**, **BAT_LOW**, **BAT_BAD**}

☝ **AVAILABILITY**

Command available only on instruments fitted with the MC01 option.

EXAMPLE

The following code will first format a memory card and then verify its status:

```
FCRD
FCRD?
```

Response message:

```
FCRD OK,RW,130048,131072,BAT_OK
```

RELATED COMMANDS

DIRECTORY

ADDITIONAL INFORMATION

Notation	
BAD	Bad card after formatting
BAT_BAD	Bad battery or no battery
BAT_LOW	Battery should be replaced
BAT_OK	Battery is in order
BLANK	Current directory empty
DIR_MISSING	No subdirectory present. The directory "LECROY1_DIR" will be automatically created with the next "store" command
NONE	No card
OK	Card is correctly formatted
RW	Read/Write authorized
WP	Write protected

MASS STORAGE — FORMAT_FLOPPY, FFLP☝

Command/Query

DESCRIPTION	The FORMAT_FLOPPY command formats a floppy disk in the Double Density or High Density format. The FORMAT_FLOPPY? query returns the status of the floppy disk.
COMMAND SYNTAX	**`Format_FLoPpy`** [<type>] <type> : = {**DD**, **HD**} If no argument is supplied, HD is used by default.
QUERY SYNTAX	**`Format_FLoPpy?`**
RESPONSE FORMAT	**`Format_FloPpy`** <floppy_status>[**,**<read/write>**,**<free_space>**,**<floppy_size>] <floppy_status> : = {**NONE**, **BAD**, **BLANK**, **DIR_MISSING**, **OK**} <read/write> : = {**WP**, **RW**} <free_space> : = A decimal number giving the number of bytes still available on the floppy. <floppy_size> : = A decimal number giving the total number of bytes on the floppy.
☝ **AVAILABILITY**	Command only available on LSA1000s fitted with the FD01 option.

EXAMPLE	The following code will first format a floppy in the Double Density (720 kB) format and then verify its status:
	`FFLP DD`
	`FFLP?`
	Response message:
	`FFLP OK,RW,728064,737280,`

RELATED COMMANDS	DIRECTORY

ADDITIONAL INFORMATION

Notation	
BAD	Bad floppy after formatting
BLANK	Current directory empty
DD	Double Density 720 kB formatted
DIR_MISSING	No subdirectory present. The directory “LECROY1_DIR” will be automatically created with the next “store” command.
HD	High Density 1.44 MB formatted
NONE	No floppy
OK	Floppy is correctly formatted
RW	Read/Write authorized
WP	Write protected

MASS STORAGE — FORMAT_HDD, FHDD☝
Command/Query

DESCRIPTION

The FORMAT_HDD command formats the removable hard disk according to the PCMIA/JEIDA standard with a DOS partition.

The FORMAT_HDD? query returns the status of the hard disk.

COMMAND SYNTAX

`Format_HDD` <type>

<type> : = {**`QUICK`**, **`FULL`**}

If no argument is supplied, QUICK will be used.

QUERY SYNTAX

`Format_HDD?`

RESPONSE FORMAT

`Format_HDD` <hdd_status>[**,**<read/write>**,**<free_space>**,** <hdd_size>]

<hdd_status> : = {**`NONE`**, **`BAD`**, **`BLANK`**, **`DIR_MISSING`**, **`OK`**}
<read/write> : = {**`WP`**, **`RW`**}
<free_space> : = A decimal number giving the number of byte still available on the hard disk
<hdd_size> : = A decimal number giving the total number of bytes on the hard disk.

☝ **AVAILABILITY**

Command only available only on instruments fitted with the HD01 option.

EXAMPLE

The following code will first format a hard disk and then verify its status:

```
FHDD
FHDD?
```

Response message:

```
FHDD OK,RW,3076096,105744896
```

RELATED COMMANDS

DIRECTORY

ADDITIONAL INFORMATION

Notation	
BAD	Bad hard disk after formatting
BLANK	Current directory empty
DIR_MISSING	No subdirectory present. The directory “LECROY1_DIR” will be automatically created with the next “store” command
NONE	No hard disk
OK	Hard disk is correctly formatted
RW	Read/Write authorized
WP	Write protected

DISPLAY FULL_SCREEN, FSCR

Command/Query

DESCRIPTION

The FULL_SCREEN command is used to control whether the currently selected grid style is displayed in normal presentation format or with a full-screen grid. In Full Screen format, the waveform display areas are enlarged to the maximum possible size.

The response to the FULL_SCREEN? query indicates whether or not the display is operating in Full Screen presentation format.

This command is included for used with programs such as ScopeExplorer.

COMMAND SYNTAX

`FullSCReen` <state>

<state> : = {**`ON`**, **`OFF`**}

QUERY SYNTAX

`FullSCReen?`

RESPONSE FORMAT

`FullSCReen` <state>

EXAMPLE

The following instruction enables the Full Screen presentation format:

```
FSCR ON
```

FUNCTION FUNCTION_RESET, FRST

Command

DESCRIPTION The FUNCTION_RESET command resets a waveform processing function. The number of sweeps will be reset to zero and the process restarted.

COMMAND SYNTAX <function> : **`Function_ReSeT`**

EXAMPLE <function> : = {**`TA`**, **`TB`**, **`TC`**, **`TD`**}

Assuming that Trace A (TA) has been defined as the summed average of Channel 1, the following instruction will restart the averaging process:

```
TA:FRST
```

RELATED COMMANDS DEFINE, INR

ACQUISITION — GAIN_MODE, GMOD

Command/Query

DESCRIPTION:	The GAIN_MODE command specifies the gain mode (gain range) of the front end.

COMMAND SYNTAX:	<channel>:Gain_MODe <mode>
	<channel> : = {**C1**, **C2**}
	<mode> := { LOW, MED, HI, AUTO }

QUERY SYNTAX:	<channel>:Gain_MODe?
RESPONSE FORMAT:	<channel>:Gain_MODe <mode>
REMARKS:	The GAIN_MODE command specifies the gain mode (gain range) of the front end. The front end has three supported gain modes, the ratios of the ranges supported by the low, medium and high gain modes are 1, 2 and 5 respectively. When changing gain modes, the current vertical gain (VDIV) setting is adapted by multiplying by the ratio of the old gain mode and the new. Thus, if the gain mode was LOW and the VDIV was 100mV/div, if the gain mode is changed to HIGH, the VDIV is multiplied by 1/5 to yield a new VDIV setting of 20mV/div.
RELATED COMMANDS:	VOLT_DIV, VOLT_RANGE

ACQUISITION GLOBAL_BWL, GBWL

Command/Query

DESCRIPTION

This turns on or off the Global Bandwidth Limit. When activated, the Bandwidth Limit applies to all channels; when deactivated, a Bandwidth Limit can be set individually for each channel (*see BWL, page 13*). The response to the GLOBAL_BWL? query indicates whether the Global Bandwidth Limit is on or off.

COMMAND SYNTAX

`Global_BWL` <mode>

<mode> : = {**`OFF`**, **`ON`**}

QUERY SYNTAX

`Global_BWL?`

RESPONSE FORMAT

`Global_BWL` <mode>

EXAMPLE

The following instruction deactivates the Global Bandwidth Limit, allowing a Bandwidth Limit to be set individually for each channel (using the BWL command syntax for individual channels):

```
GBWL OFF
```

RELATED COMMANDS

BANDWIDTH_LIMIT

DISPLAY — GRID
Command/Query

DESCRIPTION

The GRID command specifies whether the display is in single (1), dual (2), quad (4), XY or octal (8) grid mode.

The GRID? query returns the grid mode currently in use.

This command is included for used with programs such as ScopeExplorer.

COMMAND SYNTAX

`GRID` <grid>

<grid> : = {**`SINGLE`**, **`DUAL, QUAD, OCTAL, XYONLY`**☝}

QUERY SYNTAX

`GRID?`

RESPONSE FORMAT

`GRID` <grid>

☝ **AVAILABILITY**

<grid> := XYONLY available only when XY Display used.

EXAMPLE

The following instruction sets the screen display to dual grid mode:

`GRID DUAL`

RELATED COMMANDS

COLOR, INTENSITY, FULL_SCREEN

DISPLAY **HOR_MAGNIFY, HMAG**

Command/Query

DESCRIPTION

The HOR_MAGNIFY command horizontally expands the selected expansion trace by a specified factor. Magnification factors not within the range of permissible values will be rounded off to the closest legal value.

If multiple zoom is enabled, the magnification factor for all expansion traces is set to the specified factor. If the specified factor is too large for any of the expanded traces (depending on their current source), it is reduced to an acceptable value and only then applied to the traces.

The VAB bit (bit 2) in the STB register (see table on page 124) is set when a factor outside the legal range is specified.

The HOR_MAGNIFY query returns the current magnification factor for the specified expansion function.

This command is included for used with programs such as ScopeExplorer.

COMMAND SYNTAX

<exp_trace> : **`Hor_MAGnify`** <factor>
<exp_trace> : = {**`TA, TB, TC, TD`**}

<factor> : = 1 to 20000

QUERY SYNTAX

<exp_source> : **`Hor_MAGnify?`**

RESPONSE FORMAT

<exp_source> : **`Hor_MAGnify`** <factor>

EXAMPLE

The following instruction horizontally magnifies Trace A (TA) by a factor of 5:

```
TA:HMAG 5
```

RELATED COMMANDS

DUAL_ZOOM, MULTI_ZOOM

DISPLAY **HOR_POSITION, HPOS**

Command/Query

DESCRIPTION

The HOR_POSITION command horizontally positions the geometric center of the intensified zone on the source trace. Allowed positions range from division 0 through 10. If the source trace was acquired in sequence mode, horizontal shifting will only apply to a single segment at a time.

If the multiple zoom is enabled, the difference between the specified and the current horizontal position of the specified trace is applied to all expanded traces. If this would cause the horizontal position of any expanded trace to go outside the left or right screen boundaries, the difference of positions is adapted and then applied to the traces.

If the sources of expanded traces are sequence waveforms, and the multiple zoom is enabled, the difference between the specified and the current segment of the specified trace is applied to all expanded traces. If this would cause the segment of any expanded trace to go outside the range of the number of source segments, the difference is adapted and then applied to the traces.

The VAB bit (bit 2) in the STB register (see table on page 124) is set if a value outside the legal range is specified.

The HOR_POSITION query returns the position of the geometric center of the intensified zone on the source trace.

Note: Segment number 0 has the special meaning "Show All Segments Unexpanded".

This command is included for used with programs such as ScopeExplorer.

COMMAND SYNTAX

<exp_trace> : **`Hor_POSition`** <hor_position**>**,<segment>

<exp_trace> : = {**`TA, TB, TC, TD`**}
<hor_position> : = 0 to 10 DIV

<segment> : = 0 to max segments

Note 1: The suffix DIV is optional.

Note 2: The segment number is only relevant for waveforms acquired in sequence mode; it is ignored in single waveform acquisitions. When the segment number is set to 0, all segments will be shown.

QUERY SYNTAX <exp_trace> : **`Hor_POSition?`**

RESPONSE FORMAT <exp_trace> : **`Hor_POSition`** <hor_position>**[**,<segment>]

Note 3: The segment number is only given for sequence waveforms.

EXAMPLE The following instruction positions the center of the intensified zone on the trace currently viewed by Trace A (TA) at division 3:

```
TA:HPOS 3
```

RELATED COMMANDS DUAL_ZOOM, MULTI_ZOOM

MISCELLANEOUS *IDN?

Query

DESCRIPTION

The *IDN? query is used for identification purposes. The response consists of four different fields providing information on the manufacturer, the LSA1000 model, the serial number and the firmware revision level.

QUERY SYNTAX

`*IDN?`

RESPONSE FORMAT

*`IDN LECROY,`<model>`,`<serial_number>`,`<firmware_level>

<model> : =	A six- or seven-character model identifier
<serial_number> : =	A nine- or 10-digit decimal code
<firmware_level> : =	two digits giving the major release level followed by a period, then one digit giving the minor release level followed by a period and a single-digit update level (xx.y.z)

EXAMPLE

This example issues an identification request to the LSA1000:

`*IDN?`

Response message:

`*IDN LECROY,LSA1000,LSA100000000,01.0.0`

STATUS — INE

Command/Query

DESCRIPTION

The INE command sets the Internal state change Enable register (INE). This command allows one or more events in the INR register to be reflected in the INB summary message bit (bit 0) of the STB register. *For an overview of the INR defined events, refer to the table next page*.

The INE? query reads the contents of the INE register.

COMMAND SYNTAX

INE <value>

<value> : = 0 to 65535

QUERY SYNTAX

INE?

RESPONSE FORMAT

INE <value>

EXAMPLE

The following instruction allows the INB bit to be set whenever a screen dump has finished (bit 1, i.e. decimal 2), or a waveform has been acquired (bit 0, i.e. decimal 1), or both of these. Summing these two values yields the INE mask 2+1=3.

```
INE 3
```

RELATED COMMANDS

INR

STATUS — INR? Query

DESCRIPTION

The INR? query reads and clears the contents of the INternal state change Register (INR). The INR register (table below) records the completion of various internal operations and state transitions.

Internal State Register Structure (INR)

Bit	Bit Value	Description	
15...14		0	Reserved for future use
13	8192	1	Trigger is ready
12	4096	1	Pass/Fail test detected desired outcome
11	2048	1	Waveform processing has terminated in Trace D
10	1024	1	Waveform processing has terminated in Trace C
9	512	1	Waveform processing has terminated in Trace B
8	256	1	Waveform processing has terminated in Trace A
7	128	1	A memory card, floppy or hard disk exchange has been detected☝
6	64	1	Memory card, floppy or hard disk has become full in “AutoStore Fill” mode☝
5	32	0	Reserved for LeCroy use
3	8	1	A time-out has occurred in a data block transfer
2	4	1	A return to the local state is detected
0	1	1	A new signal has been acquired

QUERY SYNTAX `INR?`

RESPONSE FORMAT `INR` <state>

<state> : = 0 to 65535

☝ **AVAILABILITY** These bits only available on LSA1000s fitted with MC01, FD01, or HD01 options.

EXAMPLE The following instruction reads the contents of the INR register:

```
INR?
```

Response message:

```
INR 1026
```

i.e. waveform processing in Function C and a screen dump have both terminated.

RELATED COMMANDS ALL_STATUS, *CLS, INE

DISPLAY INTENSITY, INTS

Command/Query

DESCRIPTION

The INTENSITY command sets the intensity level of the grid or the trace/text.

The intensity level is expressed as a percentage (PCT). A level of 100 PCT corresponds to the maximum intensity whilst a level of 0 PCT sets the intensity to its minimum value.

The response to the INTENSITY? query indicates the grid and trace intensity levels.

This command is included for used with programs such as ScopeExplorer.

COMMAND SYNTAX

INTenSity GRID,< value**>, TRACE**,<value>

<value> : = 0 to 100 **[PCT]**

Note 1: Parameters are grouped in pairs. The first of the pair names the variable to be modified, whilst the second gives the new value to be assigned. Pairs may be given in any order and be restricted to those variables to be changed.

Note 2: The suffix PCT is optional.

QUERY SYNTAX

INTenSity?

RESPONSE FORMAT

INTenSity TRACE,<value>**,GRID**,<value>

EXAMPLE

The following instruction enables remote control of the intensity, and changes the grid intensity level to 75%:

```
INTS GRID,75
```

WAVEFORM TRANSFER INSPECT?, INSP?

Query

DESCRIPTION

The INSPECT? query allows the user to read parts of an acquired waveform in intelligible form. The command is based on the explanation of the format of a waveform given by the template (use the query TEMPLATE? to obtain an up-to-date copy).

Any logical block of a waveform can be inspected using this query by giving its name enclosed in quotes as the first (string) parameter (*see the template*).

The special logical block named WAVEDESC may also be inspected in more detail. By giving the name of a variable in the block WAVEDESC, enclosed in quotes as the first (string) parameter, it is possible to inspect only the actual value of that variable. *See Chapter 4 for more on INSPECT?.*

Notation	
BYTE	raw data as integers (truncated to 8 (m.s.b.[†])
FLOAT	normalized data (gain, offset applied) as floating point numbers (gives measured values in volts or units)
WORD	raw data as integers (truncated to 16 m.s.b.)

QUERY SYNTAX

<trace> : **INSPect? '**<string>**'**[**,**<data_type>]

<trace> : = {**TA**, **TB**, **TC**, **TD**, **M1**, **M2**, **M3**, **M4**, **C1**, **C2**}

<string> : = A valid name of a logical block or a valid name of a variable contained in block WAVEDESC (*see the Template*).

<data_type> : = {**BYTE**, **WORD**, **FLOAT**}

Note: The optional parameter <data_type> applies only for inspecting the data arrays. It selects the representation of the data. The default <data_type> is FLOAT.

[†] *most significant bits*

RESPONSE FORMAT	<trace> : **INSPect “**<string>**”** <string> : = A string giving name(s) and value(s) of a logical block or a variable.
EXAMPLES (GPIB)	The following instruction reads the value of the timebase at which the last waveform in Channel 1 was acquired: **C1:INSP? ‘TIMEBASE’** Response message: **C1:INSP “TIMEBASE: 500 US/DIV”** The following command reads the entire contents of the waveform descriptor block: **C1:INSP? ‘WAVEDESC’**
RELATED COMMANDS	TEMPLATE, WAVEFORM_SETUP

STATUS *IST?
Query

DESCRIPTION

The *IST? (Individual STatus) query reads the current state of the IEEE 488.1-defined “ist” local message. The “ist” individual status message is the status bit sent during a parallel poll operation.

QUERY SYNTAX

`*IST?`

RESPONSE FORMAT

`*IST` <value>

<value> : = 0 or 1

EXAMPLE

The following instruction cause the contents of the IST bit to be read:

```
*IST?
```

Response message

```
*IST 0
```

RELATED COMMANDS

*PRE

ACQUISITION — MEMORY_SIZE, MSIZ

Command/Query

DESCRIPTION

MEMORY_SIZE allows selection of the maximum memory length used for acquisition. Reducing the number of data points results in faster throughput.

The MEMORY_SIZE? query returns the current maximum memory length used to capture waveforms. When the optional suffix NUM is used with the query, the response will be returned in standard numeric format.

COMMAND SYNTAX

Memory_SIZe <size>

<size> : = {**500**, **1000**, **2500**, **5000**, **10K**, **25K**, **50K**, **100K**, **250K**, **500K**, **1M**, **2M**, **4M**, **8M**}

Note: The instrument will adapt to the closest valid <size> or numerical <value> according to available channel memory.

QUERY SYNTAX

Memory_SIZe?

RESPONSE FORMAT

Memory_SIZe <size>

EXAMPLE

The following will set the LSA1000 to acquire at most 10 000 data samples per acquisition:

MSIZ 10K

RELATED COMMANDS

TDIV, COMB?

DISPLAY — MULTI_ZOOM, MZOM

Command/Query

DESCRIPTION

By setting MULTI_ZOOM ON, the horizontal magnification and positioning controls apply to all expanded traces simultaneously. This command is useful if the contents of all expanded traces are to be examined at the same time.

The MULTI_ZOOM? query indicates whether multiple zoom is enabled or not.

Note: This command has the same effect as DUAL_ZOOM.

This command is included for used with programs such as ScopeExplorer.

COMMAND SYNTAX

`Multi_ZOoM` <mode>

<mode> : = {**`ON, OFF`**}

QUERY SYNTAX

`Multi_ZOoM?`

RESPONSE FORMAT

`Multi_ZOoM` <mode>

EXAMPLE

The following example turns the multiple zoom on:

```
MZOM ON
```

RELATED COMMANDS

HOR_MAGNIFY, HOR_POSITION, DUAL_ZOOM

ACQUISITION — OFFSET, OFST

Command/Query

DESCRIPTION

The OFFSET command allows adjustment of the vertical offset of the specified input channel.

The maximum ranges depend on the fixed sensitivity setting. If an out-of-range value is entered, the LSA1000 is set to the closest possible value and the VAB bit (bit 2) in the STB register is set.

The OFFSET? query returns the DC offset value of the specified channel.

COMMAND SYNTAX

<channel> : **OFfSeT** <offset>

<channel> : = {**C1**, **C2**}
<offset> : = *Refer to model's specifications.*

Note: The suffix V is optional.

QUERY SYNTAX

<channel> : **OFfSeT**?

RESPONSE FORMAT

<channel> : **OFfSeT** <offset>

EXAMPLE

The following command sets the offset of Channel 2 to -3 V:

```
C2:OFST -3V
```

STATUS *OPC
Command/Query

DESCRIPTION

The *OPC (OPeration Complete) command sets to true the OPC bit (bit 0) in the standard Event Status Register (ESR). This command has no other effect on the operation of the LSA1000 because the instrument starts parsing a command or query only after it has completely processed the previous command or query.

The *OPC? query always responds with the ASCII character “1” because the LSA1000 only responds to the query when the previous command has been entirely executed.

COMMAND SYNTAX `*OPC`

QUERY SYNTAX `*OPC?`

RESPONSE FORMAT `*OPC 1`

RELATED COMMANDS *WAI

MISCELLANEOUS *OPT?

Query

DESCRIPTION

The *OPT? query identifies LSA1000 options, i.e. additional firmware or hardware options. The response consists of a series of response fields listing all the installed options.

QUERY SYNTAX

`*OPT?`

RESPONSE FORMAT

`*OPT` <option_1>**,**<option_2>**,..,**<option_N>

<option_n> : = A three- or four-character ASCII string

Note: If no option is present, the character 0 will be returned

EXAMPLE

The following queries the installed options:

`*OPT?`

If, for example, the waveform processing options WP01 and WP02 are installed, the response will be returned as:

`*OPT WP01,WP02`

Response message if no options are installed:

`*OPT 0`

ADDITIONAL INFORMATION ***(not all options are necessarily available)***

Notation	
CKTR	CKTRIG Clock-Trigger-Ext. ref. Option
DDFA	Disk Drive Failure Analysis Option
DDM	Disk Drive Measurements Option
ECL	External Trigger -- ECL levels
FD01	Floppy Disk Option
HD01	Hard Disk Option
JTA	Jitter and Timing Analysis Option
ORM	Optical Recording Measurements Option
PMT	Power Measurement Tools
PRML	PRML Measurements Option
MC01	Memory Card Option
TTL	External Trigger -- TTL levels
WP01	Waveform Processing Option WP01
WP02	Waveform Processing Option WP02
WP03	Waveform Processing Option WP03

CURSOR PARAMETER_CLR, PACL
Command

DESCRIPTION

The PARAMETER_CLR command clears all the current parameters from the five-line list used in the Custom and Pass/Fail modes.

Note: This command has the same effect as the command PASS_FAIL_CONDITION, given without any arguments.

COMMAND SYNTAX

`PArameter_CLear`

RELATED COMMANDS

PARAMETER_DELETE, PARAMETER_VALUE, PASS_FAIL_CONDITION

CURSOR — PARAMETER_CUSTOM, PACU
Command/Query

DESCRIPTION

The PARAMETER_CUSTOM command controls the parameters that have customizable qualifiers, (for example, *DTLEV* or *RLEV*) and may also be used to assign any parameter for histogramming.

Note: The measured value of a parameter setup with PACU may be read using PAVA?

COMMAND SYNTAX

`PArameter_Custom` <line>,<parameter>,<qualifier>[,<qualifier>,...]

<line> : = 1 to 5

<parameter> : = {a parameter from the table below or any parameter listed in the PAVA? command}

<qualifier> : = Measurement qualifier(s) specific to each <param>. *See below.*

<param>	definition	<qualifier> list
Standard Custom Parameters *(all parameters listed are not necessarily available)*		
DDLY	delta delay	<source1>,<source2>
DTLEV	delta time at level	<source1>,<slope1>,<level1>,<source2>,<slope2>,<level2>, <hysteresis>
FLEV	fall at level	<source>,<high>,<low>
PHASE	phase difference	<source1>,<edge1>,<level1>,<source2>,<edge2>,<level2>, <hysteresis>,<angular unit>
RLEV	rise at level	<source>,<low>,<high>
TLEV	time at level	<source>,<slope>,<level>,<hysteresis>
Parameters available on instruments equipped with WP03 or DDM processing firmware		
FWXX	full width at xx% of max	<source>,<threshold>
PCTL	percentile	<source>,<threshold>
XAPK	x position at peak	<source>,<rank>

<param>	definition	<qualifier> list
Parameters available on instruments equipped with DDM processing firmware		
LBASE	local base	<source>,<hysteresis>
LBSEP	local baseline separation	<source>,<hysteresis>
LMAX	local maximum	<source>,<hysteresis>
LMIN	local minimum	<source>,<hysteresis>
LNUM	number of local events	<source>,<hysteresis>
LPP	local peak to peak	<source>,<hysteresis>
LTBE	local time between events	<source>,<hysteresis>
LTBP	local time between peaks	<source>,<hysteresis>
LTBT	local time between troughs	<source>,<hysteresis>
LTMN	local time at minima	<source>,<hysteresis>
LTMX	local time at maxima	<source>,<hysteresis>
LTOT	local time over threshold	<source>,<hysteresis>,<threshold>
LTPT	local time peak to trough	<source>,<hysteresis>
LTTP	local time trough to peak	<source>,<hysteresis>
LTUT	local time under threshold	<source>,<hysteresis>,<threshold>
NBPH	narrow band phase	<source>,<freq>
NBPW	narrow band power	<source>,<freq>
OWRITE	overwrite	<source 1>,<source 2>,<freq>
PW50	pulse width 50	<source>,<hysteresis>
PW50NEG	pulse width 50 for troughs	<source>,<hysteresis>
PW50POS	pulse width 50 for peaks	<source>,<hysteresis>
RES	resolution	<source 1>,<source 2>,<hysteresis>
TAA	track average amplitude	<source>,<hysteresis>
TAANEG	track average amplitude for troughs	<source>,<hysteresis>
TAAPOS	track average amplitude for peaks	<source>,<hysteresis>
Parameters available on instruments equipped with PRML processing firmware		
ACSN	auto correlation signal to noise	<source>,<length>
NLTS	non-linear transition shift	<source>,<length>,<delay>

Where:	<sourceN> : = {**C1**, **C2**, **TA**, **TB**, **TC**, **TD**} <slopeN> : = {**POS**, **NEG**, **FIRST**} <levelN>, <low>, <high> : = 1 to 99 if level is specified in percent (PCT), or <levelN>, <low>, <high> : = Level in <sourceN> in the units of the waveform. <delay> : = –100 PCT to 100 PCT <freq> : = 10 to 1e9 Hz (Narrow Band center frequency). <hysteresis> : = 0.01 to 8 divisions <length> : = 1e–9 to 0.001 seconds <rank> : = 1 to 100 <threshold> : = 0 to 100 percent <angular unit> = {**PCT**, **DEG**, **RAD**}
QUERY SYNTAX	**PArameter_CUstom?** <line>
RESPONSE FORMAT	**PArameter_Custom** <line>**,**<parameter>**,**<qualifier>[**,**<qualifier>**,**...]
EXAMPLE 1	**DTLEV**
Command Example	**PACU 2,DTLEV,C1,POS,345E-3,C2,NEG,-789E-3**
Query/Response Examples	**PACU? 2** returns: **PACU 2,DTLEV,C1,POS,345E-3,C2,NEG,-789E-3** **PAVA? CUST2** returns: **C2:PAVA CUST2,789 NS**
EXAMPLE 2	**DDLY**
Command Example	**PACU 2,DDLY,C1,C2**
Query/Response Examples	**PACU? 2** returns: **PACU 2,DDLY,C1,C2** **PAVA? CUST2** returns: **C2:PAVA CUST2,123 NS**

EXAMPLE 3	RLEV
Command Example	`PACU 3,RLEV,C1,2PCT,67PCT`

Query/Response Examples	`PACU? 3` returns:
	`PACU 3,RLEV,C1,2PCT,67PCT`
	`PAVA? CUST3` returns:
	`C1:PAVA CUST3,23 MS`

EXAMPLE 4	FLEV
Command Example	`PACU 3,FLEV,C1,345E-3,122E-3`

Query/Response Examples	`PACU? 3` returns:
	`PACU 3,FLEV,C1,345E-3,122E-3`
	`PAVA? CUST3` returns:
	`C1:PAVA CUST3,23 MS`

RELATED COMMANDS	PARAMETER_DELETE, PARAMETER_VALUE, PASS_FAIL_CONDITION

CURSOR — PARAMETER_DELETE, PADL

Command

DESCRIPTION

The PARAMETER_DELETE command deletes a parameter at a specified line from the list of parameters used in the Custom and Pass/Fail modes.

Notation		
1	line 1	of Custom or Pass/Fail display
2	line 2	of Custom or Pass/Fail display
3	line 3	of Custom or Pass/Fail display
4	line 4	of Custom or Pass/Fail display
5	line 5	of Custom or Pass/Fail display

COMMAND SYNTAX

`PArameter_DeLete` <line>

<line> : = {**1**, **2, 3**, **4**, **5**}

Note: This command has the same effect as the command PASS_FAIL_CONDITION <line>, given without any further arguments.

EXAMPLE

The following instruction deletes the third test condition in the list:

```
PADL 3
```

RELATED COMMANDS

PARAMETER_CLR, PARAMETER_VALUE, PASS_FAIL_CONDITION

CURSOR PARAMETER_STATISTICS?, PAST?

Query

DESCRIPTION

The PARAMETER_STATISTICS? query returns the current values of statistics for the specified pulse parameter mode and the result type, for all five lines of the pulse parameters display.

Notation	
AVG	average
CUST	custom parameters
HIGH	highest value
HPAR	horizontal standard parameters
LOW	lowest value
PARAM	parameter definition for each line
SIGMA	sigma (standard deviation)
SWEEPS	number of sweeps accumulated for each line
VPAR	vertical standard parameters

QUERY SYNTAX

PArameter_STatistics? <mode>**,** <result>

<mode> : = {**CUST**, **HPAR**, **VPAR**}

<result> : = {**AVG**, **LOW**, **HIGH**, **SIGMA**, **SWEEPS**, **PARAM**}

Note: If keyword PARAM is specified, the query returns the list of the five pairs <parameter_name>,<source>.

EXAMPLE

The following query reads the average values of the five standard vertical parameters:

PAST? VPAR, AVG

RESPONSE FORMAT

PAST VPAR, AVG, 13V, 26V, 47V, 1V, 0V

RELATED COMMANDS

PARAMETER_VALUE

CURSOR PARAMETER_VALUE?, PAVA?

Query

DESCRIPTION

The PARAMETER_VALUE query returns the current value(s) of the pulse waveform parameter(s) and mask tests for the specified trace. Traces do not need to be displayed or selected to obtain the values measured by the pulse parameters or mask tests.

Standard Parameters					
ALL	all parameters	**DUTY**	duty cycle	**OVSP**	positive overshoot
AMPL	amplitude	**FALL**	falltime	**PER**	period
AREA	area	**FALL82**	fall 80 to 20%	**PKPK**	peak-to-peak
BASE	base	**FREQ**	frequency	**PNTS**	points
CMEAN	mean for cyclic waveform	**FRST**	first point	**RISE**	risetime
CMEDI	median for cyclic waveform	**LAST**	last point	**RISE28**	rise 20 to 80%
CRMS	root mean square for cyclic part of waveform	**MAX**	maximum	**RMS**	root mean square
CSDEV	standard deviation for cyclic part of waveform	**MEAN**	mean	**SDEV**	standard deviation
CYCL	cycles	**MEDI**	median value	**TOP**	top
DLY	delay	**MIN**	minimum	**WID**	width
DUR	duration of acquisition	**OVSN**	negative overshoot		
Parameters Available on Instruments with WP03 or DDM Processing Firmware					
AVG	average of distribution	**HMEDI**	median of a histogram	**PKS**	number of peaks
DATA	data values	**HRMS**	histogram rms value	**RANGE**	range of distribution
FWHM	full width at half max	**HTOP**	histogram top value	**SIGMA**	sigma of distribution
HAMPL	histogram amplitude	**LOW**	low of distribution	**TOTP**	total population
HBASE	histogram base	**MAXP**	maximum population		
HIGH	high of histogram	**MODE**	mode of distribution		

Custom Parameters Defined using PARAMETER_CUSTOM Command[‡]				
CUST1	CUST2	CUST3	CUST4	CUST5

[‡] *The numbers in the terms CUST1, CUST2, CUST3, CUST4 and CUST5 refer to the line numbers of the selected custom parameters.*

Parameter Computation States			
AV	averaged over several (up to 100) periods	**OF**	signal partially in overflow
GT	greater than given value	**OK**	deemed to be determined without problem
IV	invalid value (insufficient data provided)	**OU**	signal partially in overflow and underflow
LT	less than given value	**PT**	window has been period truncated
NP	no pulse waveform	**UF**	signal partially in underflow
Mask Test Names			
ALL_IN	all points of waveform inside mask (TRUE = 1, FALSE = 0)	**SOME_IN**	some points of waveform inside mask (TRUE = 1, FALSE = 0)
ALL_OUT	all points of waveform outside mask (TRUE = 1, FALSE = 0)	**SOME_OUT**	some points of waveform outside mask (TRUE = 1, FALSE = 0)

QUERY SYNTAX

<trace> : **PArameter_VAlue?** [<parameter>,...,<parameter>]
<trace> : = {**TA**, **TB**, **TC**, **TD**, **C1**, **C2**}
<parameter> : = *See table of parameter names, previous page*.

Alternative forms of query for mask tests:
<trace> : **PArameter_VAlue?** <old_mask_test>
<trace> : **PArameter_VAlue?** <mask_test>, <mask>
<mask_test> : = {**ALL_IN**, **SOME_IN**, **ALL_OUT**, **SOME_OUT**}
<mask> : = {**TA**, **TB**, **TC**, **TD**}

RESPONSE FORMAT

<trace> : **PArameter_VAlue** <parameter>**,**<value>**,** <state> [**,**...**,**<parameter>**,**<value>**,**<state>]
<value> : = A decimal numeric value
<state> : = {**OK**, **AV**, **PT**, **IV**, **NP**, **GT**, **LT**, **OF**, **UF**, **OU**}

Note: If <parameter> is not specified, or is equal to ALL, all the standard voltage and standard time parameters followed by their values and states are returned.

EXAMPLE

The following query reads the risetime of Trace B (TB):

```
TB:PAVA? RISE
```

Response message:

```
TB:PAVA RISE,3.6E-9S,OK
```

RELATED COMMANDS

CURSOR_MEASURE, CURSOR_SET, PARAMETER_CUSTOM, PARAMETER_STATISTICS

CURSOR PASS_FAIL_CONDITION, PFCO
Command/Query

DESCRIPTION

The PASS_FAIL_CONDITION command adds a Pass/Fail test condition or a custom parameter at the specified line on the Pass/Fail or Custom Parameter display.

The PASS_FAIL_CONDITION? query indicates the current Pass/Fail test setup or the current selection of custom parameters at the specified line.

Note 1: Up to five test conditions (or custom parameters) can be specified at five different display lines on the screen. The command PASS_FAIL_CONDITION deals with one line at a time.

Notation			
GT	greater than	**LT**	lower than

COMMAND SYNTAX

Pass_Fail_Condition
[<line>**,**<trace>**,**<parameter>[**,**<rel_op> [,<ref_value>]]]

<line> : = {**1**,**2**,**3**,**4**,**5**}

<trace> : = {**TA**, **TB**, **TC**, **TD**, **C1**, **C2**}

<parameter> : = *See tables of parameter names on pages 92 and 98.*

<rel_op> : = {**GT**, **LT**}

<ref_value> : = –1e15 to +1e15

Note 2: The PFCO command with no arguments (i.e. “PFCO”) deletes all conditions. The PFCO command with a single argument (i.e. “PFCO <line>”) deletes the condition at <line>.

Note 3: Old mask test keywords ALLI and ANYO imply testing of <trace> against the mask waveform TD. Old mask test keywords INSIDE and OUTSIDE are equivalent to ALL_IN and SOME_OUT; they are only supported for compatibility with former versions.

Alternative form of command for mask tests:

Pass_Fail_COndition [<line>**,**<trace>**,**<mask_test>**,**<mask>]

<mask_test> : = {**ALL_IN**, **SOME_IN**, **ALL_OUT**, **SOME_OUT**}

<mask> : = {**TA**, **TB**, **TC**, **TD**}

QUERY SYNTAX	**`PFCO?`** <line>
RESPONSE FORMAT	**`PFCO`** <line>**,**<trace>**,**<parameter>**,**<rel_op>**,**<ref_value> Alternative form of response for mask tests: **`PFCO`** <line>**,**<trace>**,**<mask_test>**,**<mask>
EXAMPLE	The following instruction sets the first test condition in the list to be “frequency on Channel 1 lower than 10 kHz”: **`PFCO 1,C1,FREQ,LT,10000`**
RELATED COMMANDS	CURSOR_MEASURE, CURSOR_SET, PASS_FAIL_COUNTER, PASS_FAIL_DO, PASS_FAIL_MASK, PARAMETER_VALUE

CURSOR **PASS_FAIL_COUNTER, PFCT**
Command/Query

DESCRIPTION

The PASS_FAIL_COUNTER command resets the Passed/Failed acquisitions counters. The PASS_FAIL_COUNTER? query returns the current counts.

COMMAND SYNTAX

`Pass_Fail_CounTer`

QUERY SYNTAX

`Pass_Fail_CounTer?`

RESPONSE FORMAT

`Pass_Fail_CounTer` <pass/fail>**`,`**<value>**`,OF,`**<value>

<value> : = 0 to 999999

<pass/fail> : = {**`PASS`**, **`FAIL`**}

EXAMPLE

The following query reads the counters:

`PFCT?`

Response message:

`PFCT PASS, 8, OF, 9`

RELATED COMMANDS

CURSOR_MEASURE, CURSOR_SET, PASS_FAIL_DO, PASS_FAIL_MASK, PARAMETER_VALUE

CURSOR — PASS_FAIL_DO, PFDO

Command/Query

DESCRIPTION

The PASS_FAIL_DO command defines the desired outcome and the actions that have to be performed by the LSA1000 after a Pass/Fail test. The PASS_FAIL_DO? query indicates which actions are currently selected.

Notation	
BEEP☝	emit a beep
PULS☝	emit a pulse on the CAL connector
SCDP	make a hard copy
STO	store in memory or on storage media
STOP	stop acquisition

COMMAND SYNTAX

Pass_Fail_DO [<outcome>[,<act>[,<act>...]]]

<outcome> : = {**PASS**,**FAIL**}
<act> : = {**STOP**, **SCDP**, **STO**}

Note 2: The PFDO command with no arguments (i.e. "PFDO") deletes all actions.

Note 3: The STO command performs the store operation.

Note 4: After every pass or fail detected, the instrument sets the INR bit 12.

QUERY SYNTAX

Pass_Fail_DO?

RESPONSE FORMAT

`Pass_Fail_DO` [<pass_fail>[**,**<act>[**,**<act>...]]]

☝ **AVAILABILITY**

The BEEP command is accepted only on models equipped with the CLBZ hardware option.

The PULS command is accepted only on models equipped with the CKIO software option.

EXAMPLE

This following instruction forces the LSA1000 to stop acquiring when the test passes:

```
PFDO PASS,STOP
```

RELATED COMMANDS

BUZZER, CURSOR_MEASURE, CURSOR_SET, INR, PARAMETER_VALUE, PASS_FAIL_COUNTER, PASS_FAIL_MASK

CURSOR PASS_FAIL_MASK, PFMS

Command

DESCRIPTION

The PASS_FAIL_MASK command generates a tolerance mask around a chosen trace and stores the mask in the selected memory.

COMMAND SYNTAX

`Pass_Fail_MaSk` [<trace>[**,**<htol>[**,**<vtol>[**,**<mask>]]]]

<trace> : = {**TA**, **TB**, **TC**, **TD**, **M1**, **M2**, **M3**, **M4**, **C1**, **C2**}
<htol> : = 0.0 to 5.0
<vtol> : = 0.0 to 4.0
<mask> : = {**M1**, **M2**, **M3**, **M4**}

Note: if any arguments are missing, the previous settings will be used.

The alternative form of command:

`Pass_Fail_MaSk INVT` [**,**<mask>]

inverts the mask in the selected mask memory. If <mask> is missing, M4 is implied.

EXAMPLE

The following instruction generates a tolerance mask around the Channel 1 trace and stores it in M2:

```
PASS_FAIL_MASK C1,0.2,0.3,M2
```

RELATED COMMANDS

PASS_FAIL_DO, PARAMETER_VALUE

CURSOR PASS_FAIL_STATUS?, PFST?

Query

DESCRIPTION

The PASS_FAIL_STATUS query returns the status of the pass/fail test for a given line number.

QUERY SYNTAX

`Pass_Fail_STatus?` <line>

<line> : = {**1**, **2**, **3**, **4**, **5**}

RESPONSE FORMAT

`Pass_Fail_STatus` <line>**,**<state>

<state> : = {**TRUE**, **FALSE**}

EXAMPLE

The following queries the state of the pass/fail test condition specified for line 3.

```
PFST? 3
```

RELATED COMMANDS

PASS_FAIL_DO, PASS_FAIL_CONDITION, PARAMETER_VALUE

CURSOR PER_CURSOR_SET, PECS
Command/Query

DESCRIPTION

The PER_CURSOR_SET command allows the user to position any one of the six independent cursors at a given location. The position of the cursor can be modified or queried.

The PER_CURSOR_SET? query indicates the current position of the cursor(s).

The vertical cursor positions are the same as those controlled by the CURSOR_SET command.

Notation			
HABS	horizontal absolute	**VABS**	vertical absolute
HDIF	horizontal difference	**VDIF**	vertical difference
HREF	horizontal reference	**VREF**	vertical reference

COMMAND SYNTAX

<trace> : **PEr_Cursor_Set** <cursor>**,** <position>[**,**<cursor>**,**<position>**,**...**,**<cursor>,<position>

trace> : = {**TA**, **TB**, **TC**, **TD**, **C1**, **C2**}
<cursor> : = {**HABS**, **HDIF**, **HREF**, **VABS**, **VDIF**, **VREF**}
<position> : = 0 to 10 DIV (horizontal), –29.5 to 29.5 DIV (vertical)

Note 1: The suffix DIV is optional.

Note 2: Parameters are grouped in pairs. The first of the pair names the variable to be modified, whilst the second gives the new value to be assigned. Pairs may be in any order and be restricted to those variables to be changed.

QUERY SYNTAX

<trace> **: PEr_Cursor_Set?** <cursor>[**,**<cursor**,**...**,**<cursor>]

<cursor> : = {**HABS**, **HDIF**, **HREF**, **VABS**, **VDIF**, **VREF**, **ALL**}

Note 3: If <cursor> is not specified, ALL will be assumed. If the position of a cursor cannot be determined in a particular situation, its position will be indicated as UNDEF.

RESPONSE FORMAT

`PEr_Cursor_Set` <cursor>,<position>[,<cursor>,<position>,...,<cursor>,<position>

EXAMPLE

The following code positions the HREF and HDIF cursors at +2.6 DIV and +7.4 DIV respectively, using Channel 2 as a reference:

```
C2:PECS HREF,2.6 DIV,HDIF,7.4DIV
```

RELATED COMMANDS

CURSOR_MEASURE, CURSOR_SET, PERSIST, PER_CURSOR_VALUE,

CURSOR PER_CURSOR_VALUE?, PECV?

Query

DESCRIPTION

The PER_CURSOR_VALUE? query returns the values measured by the cursors specified below while in Persistence Mode.

Notation			
HABS	horizontal absolute	**VABS**	vertical absolute
HREL	horizontal relative	**VREL**	vertical relative

QUERY SYNTAX

<trace> : **PEr_Cursor_Value?** <cursor>[**,**<cursor>**,**...**,**<cursor>]

<trace> : = {**TA**, **TB**, **TC**, **TD**, **C1**, **C2**}
<cursor> : = {**HABS**, **HREL**, **VABS**, **VREL**, **ALL**}

Note: If <cursor> is not specified, ALL will be assumed.

RESPONSE FORMAT

<trace> **: PEr_Cursor_Value** <cursor>**,** <value>[**,**<cursor>**,**<value>**,**...**,**<cursor>**,**<value>]

EXAMPLE

The following code returns the value measured with the vertical relative cursor on Channel 1:

C1:PECV? VREL

Response message:

C1:PECV VREL,56 MV

RELATED COMMANDS

CURSOR_MEASURE, PERSIST, PER_CURSOR_SET

DISPLAY PERSIST, PERS

Command/Query

DESCRIPTION

The PERSIST command enables or disables the persistence display mode.

This command is included for used with programs such as ScopeExplorer.

COMMAND SYNTAX

`PERSist` <mode>

<mode> : = {**`ON, OFF`**}

QUERY SYNTAX

`PERSist?`

RESPONSE FORMAT

`PERSist` <mode>

EXAMPLE

The following code turns the persistence display ON:

```
PERS ON
```

RELATED COMMANDS

PERSIST_COLOR, PERSIST_LAST, PERSIST_SAT, PERSIST_SETUP

DISPLAY PERSIST_COLOR, PECL

Command/Query

DESCRIPTION

The PERSIST_COLOR command controls the color rendering method of persistence traces.

The response to the PERSIST_COLOR? query indicates the color rendering method, Analog Persistence or Color Graded Persistence.

This command is included for used with programs such as ScopeExplorer.

COMMAND SYNTAX

PErsist_CoLor <state>

<state> : = {**ANALOG**, **COLOR_GRADED**}

QUERY SYNTAX

PErsist_CoLor?

RESPONSE FORMAT

PErsist_CoLor <state>

EXAMPLE

The following instruction sets the persistence trace color to an intensity-graded range of the selected trace color:

```
PECL ANALOG
```

RELATED COMMANDS

COLOR, COLOR_SCHEME, PERSIST, PERSIST_LAST, PERSIST_SAT, PERSIST_SETUP

DISPLAY PERSIST_LAST, PELT

Command/Query

DESCRIPTION

The PERSIST_LAST command controls whether or not the last trace drawn in a persistence data map is shown.

The response to the PERSIST_LAST? query indicates whether the last trace is shown within its persistence data map.

This command is included for used with programs such as ScopeExplorer.

COMMAND SYNTAX

PErsist_LasT <state>

<state> : = {**ON**, **OFF**}

QUERY SYNTAX

PErsist_LasT?

RESPONSE FORMAT

PErsist_LasT <state>

EXAMPLE

The following instruction ensures the last trace is visible within its persistence data map:

```
PELT ON
```

RELATED COMMANDS

PERSIST, PERSIST_COLOR, PERSIST_SAT, PERSIST_SETUP

DISPLAY PERSIST_SAT, PESA

Command/Query

DESCRIPTION

The PERSIST_SAT command sets the level at which the color spectrum of the persistence display is saturated.

The level is specified in terms of percentage (PCT) of the total persistence data map population. A level of 100 PCT corresponds to the color spectrum being spread across the entire depth of the persistence data map. At lower values, the spectrum will saturate (brightest value) at the specified percentage value. The PCT is optional.

The response to the PERSIST_SAT? query indicates the saturation level of the persistence data maps.

This command is included for used with programs such as ScopeExplorer.

COMMAND SYNTAX

`PErsist_SAt` <trace>,<value> [<trace>,<value>]

<trace> : = { **C1**, **C2**, **C3**, **C4**, **TA**, **TB**, **TC**, **TD**, **ALL**}

<value> : = 0 to 100 PCT

Note: The suffix PCT is optional.

QUERY SYNTAX

`PErsist_SAt?`

RESPONSE FORMAT

`PErsist_SAt` <trace>,<value>

EXAMPLE

The following instruction sets the saturation level of the persistence data map for channel 3 to be 60%, i.e. 60% of the data points will be displayed with the color spectrum, with the remaining 40% saturated in the brightest color:

`PESA C3,60`

RELATED COMMANDS

PERSIST, PERSIST_COLOR, PERSIST_PERS, PERSIST_SETUP

DISPLAY PERSIST_SETUP, PESU

Command/Query

DESCRIPTION

The PERSIST_SETUP command selects the persistence duration of the display, in seconds, in persistence mode. In addition, the persistence can be set either to all traces or only the top two on the screen.

The PERSIST_SETUP? query indicates the current status of the persistence.

This command is included for used with programs such as ScopeExplorer.

COMMAND SYNTAX

`PErsist_SetUp` <time>,<mode>

<time> : = {**`0.5`**, **`1`**, **`2`**, **`5`**, **`10`**, **`20`**, **`infinite`**}

<mode> : = {**`TOP2`**, **`ALL`**}

QUERY SYNTAX

`PErsist_SetUp?`

RESPONSE FORMAT

`PErsist_SetUp` <time>,<mode>

EXAMPLE

The following instruction sets the variable persistence at 10 seconds on the top two traces:

```
PESU 20,TOP2
```

RELATED COMMANDS

PERSIST, PERSIST_COLOR, PERSIST_PERS, PERSIST_SAT

STATUS ***PRE**

Command/Query

DESCRIPTION

The *PRE command sets the PaRallel poll Enable register (PRE). The lowest eight bits of the Parallel Poll Register (PPR) are composed of the STB bits. The *PRE command allows the user to specify which bit(s) of the parallel poll register will affect the 'ist' individual status bit.

The *PRE? query reads the contents of the PRE register. The response is a decimal number which corresponds to the binary sum of the register bits.

COMMAND SYNTAX

PRE <value>

<value> : = 0 to 65 535

QUERY SYNTAX

***PRE?**

RESPONSE FORMAT

***PRE** <value>

EXAMPLE

The following instruction will cause the 'ist' status bit to become 1 as soon as the MAV bit (bit 4 of STB, i.e. decimal 16) is set. This yields the PRE value 16.

```
*PRE 16
```

RELATED COMMANDS

*IST

WAVEFORM TRANSFER — RECALL, REC

Command

DESCRIPTION

The RECALL command recalls a waveform file from the current directory on mass storage into any or all of the internal memories M1 to M4. *Note that only waveforms stored in BINARY format can be recalled.*

COMMAND SYNTAX

<memory> : **RECall DISK,**<device>**,**FILE**,'**<filename>**'**

<memory> : = {**M1**, **M2**, **M3**, **M4**, **ALL**}

<device> : = {**CARD**, **FLPY**, **HDD**}

<filename> : = An alphanumeric string of up to eight characters, followed by a dot and an extension of up to three digits.

AVAILABILITY

This command is available only when the FD01, HD01 or MC01 option is fitted.

<device> : **CARD** only available when MC01 Option is fitted.

<device> : **FLPY** only available when FD01 Option is fitted.

<device> : **HDD** only available when HD01 Option is fitted.

EXAMPLE

The following recalls a waveform file called "SC1.001" from the memory card into Memory M1:

```
M1:REC DISK,CARD,FILE,'SC1.001'
```

RELATED COMMANDS

FUNCTION_STATE, STORE, INR?

ACQUISITION — REFERENCE_CLOCK, RCLK

Command/Query

DESCRIPTION

The REFERENCE_CLOCK selects the system clock source, allowing the LSA1000 to be phase-synchronized to an external reference clock input.

COMMAND SYNTAX

Reference_CLocK <state>

<state>: = {**INT**, **EXT**}

QUERY SYNTAX

Reference_CLocK?

RESPONSE FORMAT

Reference_CLocK <state>

EXAMPLE

The following command sets the LSA1000 to use the external reference clock.

```
RCLK EXT
```

RELATED COMMANDS

*CAL, *RCL

SAVE/RECALL SETUP — *RST Command

DESCRIPTION

The *RST command initiates a device, recalls the default setup, and causes a calibration to be performed.

COMMAND SYNTAX

***RST**

EXAMPLE

This example resets the LSA1000:

***RST**

RELATED COMMANDS

*CAL, *RCL

ACQUISITION — SEQUENCE, SEQ

Command/Query

DESCRIPTION

The SEQUENCE command sets the conditions for the sequence mode acquisition. The response to the SEQUENCE? query gives the conditions for the sequence mode acquisition. The argument <max_size> can be expressed either as numeric fixed point, exponential or using standard suffixes. When the optional suffix NUM is used with the query, the response will be returned in standard numeric format.

COMMAND SYNTAX

SEQuence <mode>[**,**<segments>[**,**<max_size>]]

<mode> : = {**OFF**, **ON**, **WRAP**}

<segments> : = the right-hand column in the table below:

Max. memory length per channel	Max. number of segments
10 k	50
25 k	50
50 k	200
100 k	500
200 k	500
250 k	500
500 k	2000
1 M	2000
2 M	2000
4 M	2000

<max_size> : = {**50**, **100**, **250**, **500**, **1000**, **2500**, **5K**, **10K**, **25K**, **50K**, **100K**, **250K**, **500K**, **1M**}

Or, alternatively, in standard numeric format:

= {…**10e+3**, **10.0e+3**,…**11e+3**,…}, for example.

Note: The instrument will adapt the requested <max_size> to the closest valid value.

QUERY SYNTAX

SEQuence? [**NUM**]

RESPONSE FORMAT

SEQuence <mode>**,**<segments>**,**<max_size>

<mode> : = {**ON**, **OFF**}

EXAMPLE (GPIB)

The following sets the segment count to 43, the maximum segment size to 250 samples, and turns the sequence mode ON:

```
SEQ ON,43,250
```

RELATED COMMANDS

TRIG_MODE

STATUS ***SRE**

Command/Query

DESCRIPTION

The *SRE command sets the Service Request Enable register (SRE). This command allows the user to specify which summary message bit(s) in the STB register will generate a service request. *Refer to the table on page 124 for an overview of the available summary messages.*

A summary message bit is enabled by writing a ‘1’ into the corresponding bit location. Conversely, writing a ‘0’ into a given bit location prevents the associated event from generating a service request (SRQ). Clearing the SRE register disables SRQ interrupts.

The *SRE? query returns a value that, when converted to a binary number, represents the bit settings of the SRE register. Note that bit 6 (MSS) cannot be set and its returned value is always zero.

COMMAND SYNTAX

***SRE** <value>

<value> : = 0 to 255

QUERY SYNTAX

***SRE?**

RESPONSE FORMAT

***SRE** <value>

EXAMPLE

The following instruction allows an SRQ to be generated as soon as the MAV summary bit (bit 4, i.e. decimal 16) or the INB summary bit (bit 0, i.e. decimal 1) in the STB register, or both, are set. Summing these two values yields the SRE mask 16+1 = 17.

```
*SRE 17
```

STATUS *STB?

Query

DESCRIPTION

The *STB? query reads the contents of the 488.1 defined status register (STB), and the Master Summary Status (MSS). The response represents the values of bits 0 to 5 and 7 of the Status Byte register and the MSS summary message.

The response to a *STB? query is identical to the response of a serial poll except that the MSS summary message appears in bit 6 in place of the RQS message. *Refer to the table on page 124 for further details of the status register structure.*

QUERY SYNTAX

```
*STB?
```

RESPONSE FORMAT

*STB <value>

<value> : = 0 to 255

EXAMPLE

The following reads the status byte register:

```
*STB?
```

Response message:

```
*STB 0
```

RELATED COMMANDS

ALL_STATUS, *CLS, *PRE, *SRE

ADDITIONAL INFORMATION

Status Byte Register (STB)					
Bit	**Bit Value**	**Bit Name**	**Description**		**Note**
7	128	DIO7	0	reserved for future use	
6	64	MSS/RQS MSS=1 RQS=1		 at least 1 bit in STB masked by SRE is 1 service is requested	(1) (2)
5	32	ESB	1	an ESR enabled event has occurred	(3)
4	16	MAV	1	output queue is not empty	(4)
3	8	DIO3	0	reserved	
2	4	VAB	1	a command data value has been adapted	(5)
1	2	DIO1	0	reserved	
0	1	INB	1	an enabled INternal state change has occurred	(6)

Notes

(1) The Master Summary Status (MSS) indicates that the instrument requests service, whilst the Service Request status — when set — specifies that the LSA1000 issued a service request. Bit position 6 depends on the polling method:

*Bit 6 = MSS if an *STB? query is received*

= RQS if serial polling is conducted

(2) Example: If SRE=10 and STB=10 then MSS=1. If SRE=010 and STB=100 then MSS=0.

(3) The Event Status Bit (ESB) indicates whether or not one or more of the enabled IEEE 488.2 events have occurred since the last reading or clearing of the Standard Event Status Register (ESR). ESB is set if an enabled event becomes true (1).

(4) The Message AVailable bit (MAV) indicates whether or not the Output queue is empty. The MAV summary bit is set true (1) whenever a data byte resides in the Output queue.

(5) The Value Adapted Bit (VAB) is set true (1) whenever a data value in a command has been adapted to the nearest legal value. For instance, the VAB bit would be set if the timebase is redefined as 2.5 µs/div since the adapted value is 2 µs/div.

(6) The INternal state Bit (INB) is set true (1) whenever certain enabled internal states are entered. For further information, refer to the INR query.

ACQUISITION — STOP Command

DESCRIPTION

The STOP command immediately stops the acquisition of a signal. If the trigger mode is AUTO or NORM, it will change to trigger mode STOPPED to prevent further acquisition.

COMMAND SYNTAX

```
STOP
```

EXAMPLE

The following stops the acquisition process:

```
STOP
```

RELATED COMMANDS

ARM_ACQUISITION, TRIG_MODE, WAIT

WAVEFORM TRANSFER STORE, STO
Command

DESCRIPTION

The STORE command stores the contents of the specified trace into one of the internal memories M1 to M4.

COMMAND SYNTAX

`STOre` [<trace>,<dest>]

<trace> : = {**`TA`**, **`TB`**, **`TC`**, **`TD`**, **`C1`**, **`C2`**, **`ALL_DISPLAYED`**}
<dest> : = {**`M1`**, **`M2`**, **`M3`**, **`M4`**}

Note: If the STORE command is sent without any argument, all traces currently enabled in the Store Setup will be stored. This setup can be modified using the STORE_SETUP command.

EXAMPLE

The following command stores the contents of Trace A (TA) into Memory 1 (M1):

```
STO TA,M1
```

The following command executes the storage operation currently defined in the Storage Setup (see command STORE_SETUP, page 127):

```
STO
```

RELATED COMMANDS

STORE_SETUP, RECALL

WAVEFORM TRANSFER — STORE_SETUP, STST

Command/Query

DESCRIPTION

The STORE_SETUP command controls the way in which traces will be stored. A single trace or all displayed traces may be enabled for storage. This applies to auto-storing or to the STORE, STO command.

The STORE_SETUP? query returns the current mode of operation of Autostore, the current trace selection, and the current destination.

Note that only waveforms stored in BINARY format can be recalled.

COMMAND SYNTAX

STore_SeTup [<trace>, <dest>] [, **AUTO**, <mode>] [, **FORMAT**, <type>]

<trace> : = {**TA**, **TB**, **TC**, **TD**, **C1**, **C2**, **ALL_DISPLAYED**}

<dest> : = {**M1**, **M2**, **M3**, **M4**}

<mode> : = {**OFF**, **WRAP**, **FILL**}

<type> : {**BINARY**, **SPREADSHEET**, **MATHCAD**, **MATLAB** }

QUERY SYNTAX

STore_SeTup?

RESPONSE FORMAT

STore_SeTup <trace>**,**<dest>**,AUTO,**<mode>

EXAMPLE

The following selects Channel 1 to be stored and enables an “autostore” to the M1 memory until no more space is left in the memory.

STST C1, M1, AUTO,FILL

RELATED COMMANDS

STORE, INR

WAVEFORM TRANSFER — STORE_TEMPLATE, STTM☝ Command

DESCRIPTION

The STORE_TEMPLATE command stores the instrument's waveform template on a mass-storage device. A filename is automatically generated in the form of "LECROYvv.TPL" where "vv" is the two-digit revision number.

Note: For revision 2.1, for example, the file name generated will be LECROY21.TPL.

Refer to Chapter 4 for more on the waveform template, and Appendix B for a copy of the template itself.

COMMAND SYNTAX

STore_TeMplate DISK,<device>

<device> : = {**CARD**☝, **FLPY**☝, **HDD**☝}

☝ **AVAILABILITY**

This command is available only when the FD01, HD01 or MC01 option is fitted.

<device> : CARD only available when MC01 option is fitted.
<device> : FLPY only available when FD01 option is fitted
<device> : HDD only available when HD01 option is fitted.

EXAMPLE

The following code stores the current waveform template on the memory card for future reference:

```
STTM DISK, CARD
```

RELATED COMMANDS

TEMPLATE

WAVEFORM TRANSFER **TEMPLATE?, TMPL?**

Query

DESCRIPTION The TEMPLATE? query 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 Chapter 4 for more on the waveform template, and Appendix A for a copy of the template itself*.

QUERY SYNTAX **`TeMPLate?`**

RESPONSE FORMAT **`TeMPLate`** **"**<template>**"**

<template> : = A variable length string detailing the structure of a waveform.

RELATED COMMANDS INSPECT

ACQUISITION TIME_DIV, TDIV

Command/Query

DESCRIPTION

The TIME_DIV command modifies the timebase setting. The new timebase setting may be specified with suffixes: NS for nanoseconds, US for microseconds, MS for milliseconds, S for seconds, or KS for kiloseconds. An out-of-range value causes the VAB bit (bit 2) in the STB register (*see table on page 124*) to be set.

The TIME_DIV? query returns the current timebase setting.

COMMAND SYNTAX

`Time_DIV` <value>

<value> : = *Refer to model's specifications.*

Note: The suffix S (seconds) is optional.

QUERY SYNTAX

`Time_DIV?`

RESPONSE FORMAT

`Time_DIV` <value>

EXAMPLE

The following sets the time base to 500 µsec/div:

```
TDIV 500US
```

The following sets the time base to 2 msec/div:

```
TDIV 0.002
```

RELATED COMMANDS

TRIG_DELAY, TRIG_MODE

DISPLAY — TRACE, TRA

Command/Query

DESCRIPTION

The TRACE command enables or disables the display of a trace. An environment error (see table on page 60) is set if an attempt is made to display more than four waveforms.

The TRACE? query indicates whether the specified trace is displayed or not.

COMMAND SYNTAX

<trace> : **TRAce** <mode>

<trace> : = {**C1**, **C2**, **TA**, **TB**, **TC**, **TD**}

<mode> : = {**ON**, **OFF**}

QUERY SYNTAX

<trace> : **TRAce?**

RESPONSE FORMAT

<trace> : **TRAce** <mode>

EXAMPLE

The following command displays Trace A (TA):

```
TA:TRA ON
```

RELATED COMMANDS

DEFINE

ACQUISITION ***TRG**
Command

DESCRIPTION

The *TRG command executes an ARM command.

*Note: The *TRG command is the equivalent of the 488.1 GET (Group Execute Trigger) message.*

COMMAND SYNTAX

```
*TRG
```

EXAMPLE

The following command enables signal acquisition:

```
TRG
```

RELATED COMMANDS

ARM_ACQUISITION, STOP, WAIT

ACQUISITION TRIG_DELAY, TRDL

Command/Query

DESCRIPTION

The TRIG_DELAY command sets the time at which the trigger is to occur with respect to the first acquired data point (displayed at the left-hand edge of the screen).

The command expects positive trigger delays to be expressed as a percentage of the full horizontal screen. This mode is called pre-trigger acquisition, as data are acquired before the trigger occurs. Negative trigger delays must be given in seconds. This mode is called post-trigger acquisition, as the data are acquired after the trigger has occurred.

If a value outside the range –10 000 div X time/div and 100% is specified, the trigger time will be set to the nearest limit and the VAB bit (bit 2) will be set in the STB register.

The response to the TRIG_DELAY? query indicates the trigger time with respect to the first acquired data point. Positive times are expressed as a percentage of the full horizontal screen and negative times in seconds.

COMMAND SYNTAX

TRig_DeLay <value>

```
<value> : =       0.00 PCT to 100.00 PCT (pretrigger)
              –20 PS to –10 MAS (post-trigger)
```

Note: The suffix is optional. For positive numbers the suffix PCT is assumed. For negative numbers the suffix S is assumed. MAS is the suffix for Ms (megaseconds), useful only for extremely large delays at very slow timebases.

QUERY SYNTAX

TRig_DeLay?

RESPONSE FORMAT

TRig_DeLay <value>

EXAMPLE

The following command sets the trigger delay to –20 s (post-trigger):

TRDL - 20S

RELATED COMMANDS

TIME_DIV, TRIG_COUPLING, TRIG_LEVEL, TRIG_LEVEL_2, TRIG_MODE, TRIG_SELECT, TRIG_SLOPE, TRIG_WINDOW

ACQUISITION — TRIG_LEVEL, TRLV

Command/Query

DESCRIPTION

The TRIG_LEVEL command adjusts the trigger level of the specified trigger source. An out-of-range value will be adjusted to the closest legal value and will cause the VAB bit (bit 2) in the STB register to be set.

The TRIG_LEVEL? query returns the current trigger level.

COMMAND SYNTAX

<trig_source> : **TRig_LeVel** <trig_level>

<trig_source> : = {**C1**, **C2**}
<trig_level> : = *Refer to model's specifications.*

Note: The suffix V is optional.

QUERY SYNTAX

<trig_source> : **TRig_LeVel?**

RESPONSE FORMAT

<trig_source> : **TRig_LeVel** <trig_level>

EXAMPLE

The following code adjusts the trigger level of Channel 2 to -3.4 V:

```
C2:TRLV -3.4V
```

RELATED COMMANDS

TRIG_COUPLING, TRIG_DELAY, TRIG_LEVEL_2, TRIG_MODE, TRIG_SELECT, TRIG_SLOPE, TRIG_WINDOW

ACQUISITION TRIG_MODE, TRMD

Command/Query

DESCRIPTION

The TRIG_MODE command specifies the trigger mode.

The TRIG_MODE? query returns the current trigger mode.

COMMAND SYNTAX

`TRig_MoDe` <mode>

<mode> : = {**`AUTO`**, **`NORM`**, **`SINGLE`**, **`STOP`**}

QUERY SYNTAX

`TRig_MoDe?`

RESPONSE FORMAT

`TRig_MoDe` <mode>

EXAMPLE

The following selects the normal mode:

```
TRMD NORM
```

RELATED COMMANDS

ARM_ACQUISITION, STOP, TRIG_SELECT, TRIG_COUPLING, TRIG_LEVEL, TRIG_SLOPE, TRIG_WINDOW

ACQUISITION **TRIG_SELECT, TRSE**
Command/Query

DESCRIPTION

The TRIG_SELECT command selects the condition that will trigger the acquisition of waveforms. Depending on the trigger type, additional parameters must be specified. These additional parameters are grouped in pairs. The first in the pair names the variable to be modified, while the second gives the new value to be assigned. Pairs may be given in any order and restricted to those variables to be changed.

The TRIG_SELECT? query returns the current trigger condition.

Trigger Notation			
EDGE	Edge	**WIND**	Window

COMMAND SYNTAX

TRig_SElect <trig_type>**,SR,**<source>**,HT,**<hold_type>

<trig_type> : = {**EDGE**, **WIND**}
<source> : = {**C1**, **C2**, **EX**}
<hold_value> : = {**OFF**} — **OFF** is the only hold type available

Note: The suffix S (seconds) is optional.

QUERY SYNTAX

TRig_SElect?

RESPONSE FORMAT

TRig_SElect <trig_type>**,SR,**<source>**,HT,OFF**

AVAILABILITY

<trig_type> : WIND not available with EX as source.

EXAMPLE

The following sets up the trigger system to trigger on Channel 1:

TRSE EDGE,SR,C1

RELATED COMMANDS

TRIG_COUPLING, TRIG_DELAY, TRIG_LEVEL, TRIG_MODE, TRIG_SLOPE, TRIG_SLOPE, TRIG_WINDOW

ACQUISITION TRIG_SLOPE, TRSL

Command/Query

DESCRIPTION

The TRIG_SLOPE command sets the trigger slope of the specified trigger source. An environment error (*see table on page 60*) will be generated when an incompatible TRSL order is received — for example, if the current trigger type is EDGE, a slope of WIN or WOUT may not be specified.

The TRIG_SLOPE? query returns the trigger slope of the selected source.

COMMAND SYNTAX

<trig_source> : **TRig_SLope** <trig_slope>

<trig_source> : = {**C1**, **C2**, **EX**}

<trig_slope> : = {**NEG**, **POS**, **WOUT**☝, **WIN**☝}

QUERY SYNTAX

<trig_source> : **TRig_SLope?**

RESPONSE FORMAT

<trig_source> **: TRig_SLope** <trig_slope>

☝ **AVAILABILITY**

<trig_source> : = WOUT and WIN not available with EX as source.

EXAMPLE

The following sets the trigger slope of Channel 2 to negative:

```
C2:TRSL NEG
```

RELATED COMMANDS

TRIG_COUPLING, TRIG_DELAY, TRIG_LEVEL, TRIG_MODE, TRIG_SELECT, TRIG_SLOPE, TRIG_WINDOW

ACQUISITION — TRIG_WINDOW, TRWI

Command/Query

DESCRIPTION

The TRIG_WINDOW command sets the window amplitude in volts on the specified trigger source. The window is centered around the Edge trigger level.

The TRIG_WINDOW? query returns the current window amplitude.

COMMAND SYNTAX

<trig_source>**:TRig_WIndow** <value>

<source>: = {**C1**, **C2**}

<value> : = 0 to 1 V (maximum range)

Note: The suffix V is optional.

QUERY SYNTAX

<trig_source> : **TRig_WIndow?**

RESPONSE FORMAT

<trig_source> : **TRig_WIndow** <trig_level>

EXAMPLE

The following command adjusts the window size to +0.5 V on Channel 1:

```
C1:TRWI 0.5V
```

RELATED COMMANDS

TRIG_COUPLING, TRIG_DELAY, TRIG_LEVEL, TRIG_MODE, TRIG_SELECT, TRIG_SLOPE

MISCELLANEOUS ***TST?**

Query

DESCRIPTION

The *TST? query performs an internal self-test, the response indicating whether the self-test has detected any errors. The self-test includes testing the hardware of all channels, the timebase and the trigger circuits.

Hardware failures are identified by a unique binary code in the returned <status> number. A “0” response indicates that no failures occurred.

QUERY SYNTAX

***TST?**

RESPONSE FORMAT

***TST** <status>

<status> : = 0   self-test successful

EXAMPLE

The following causes a self-test to be performed:

***TST?**

Response message (if no failure):

***TST 0**

RELATED COMMANDS

*CAL

DISPLAY — VERT_MAGNIFY, VMAG

Command/Query

DESCRIPTION

The VERT_MAGNIFY command vertically expands the specified trace. The command is executed even if the trace is not displayed.

The maximum magnification allowed depends on the number of significant bits associated with the data of the trace.

The VERT_MAGNIFY? query returns the magnification factor of the specified trace.

This command is included for used with programs such as ScopeExplorer.

COMMAND SYNTAX

<trace> : **`Vert_MAGnify`** <factor>

<trace> : = {**TA**, **TB**, **TC**, **TD**}

<factor> : = 4.0E-3 to 50 (maximum)

QUERY SYNTAX

<trace> : **`Vert_MAGnify?`**

RESPONSE FORMAT

<trace> : **`Vert_MAGnify`** <factor>

EXAMPLE

The following command enlarges the vertical amplitude of Trace A by a factor of 3.45 with respect to its original amplitude:

```
TA:VMAG 3.45
```

RELATED COMMANDS

VERT_POSITION

DISPLAY VERT_POSITION, VPOS

Command/Query

DESCRIPTION

The VERT_POSITION command adjusts the vertical position of the specified trace on the screen. It does not affect the original offset value obtained at acquisition time.

The VERT_POSITION? query returns the current vertical position of the specified trace.

This command is included for used with programs such as ScopeExplorer.

COMMAND SYNTAX

<trace> : **`Vert_POSITION`** <display_offset>

<trace> : = {**`TA`**, **`TB`**, **`TC`**, **`TD`**}

<display_offset> : = -5900 to +5900 DIV

Note: The suffix DIV is optional. The limits depend on the current magnification factor, the number of grids on the display, and the initial position of the trace.

QUERY SYNTAX

<trace> : **`Vert_POSition?`**

RESPONSE FORMAT

<trace> : **`Vert_POSITION`** <display_offset>

EXAMPLE

The following shifts Trace A (TA) upwards by +3 divisions relative to the position at the time of acquisition:

```
TA:VPOS 3DIV
```

RELATED COMMANDS

VERT_MAGNIFY

ACQUISITION VOLT_DIV, VDIV

Command/Query

DESCRIPTION

The VOLT_DIV command sets the vertical sensitivity in Volts/div. The VAB bit (bit 2) in the STB register (*see table on page 124*) is set if an out-of-range value is entered.

The VOLT_DIV query returns the vertical sensitivity of the specified channel.

COMMAND SYNTAX

<channel**>** : **`Volt_DIV`** <v_gain>

<channel> : = {**`C1`**, **`C2`**}
<v_gain> : = *Refer to model's specifications.*

Note: The suffix V is optional.

QUERY SYNTAX

<channel> : **`Volt_DIV?`**

RESPONSE FORMAT

<channel> : **`Volt_DIV`** <v_gain>

EXAMPLE

The following command sets the vertical sensitivity of channel 1 to 50 mV/div:

```
C1:VDIV 50MV
```

RELATED COMMANDS

GAIN_MODE, VOLT_RANGE

ACQUISITION **VOLT_RANGE, VRNG**
Command/Query

DESCRIPTION

The VOLT_RANGE command sets the full-scale range in volts. The VAB bit (bit 2) in the STB register (*see table on page 124*) is set if an out-of-range value is entered.

The VOLT_RANGE query returns the full-scale range of the specified channel.

COMMAND SYNTAX

<channel> : **Volt_RaNGe** <v_range>

<channel> : = {**C1**, **C2**}
<v_range> : = *Refer to model specifications.*

Note: The suffix V is optional.

QUERY SYNTAX

<channel> : **Volt_RaNGe?**

RESPONSE FORMAT

<channel> : **Volt_RaNGe** <v_range>

AVAILABILITY

Command is only available if model supports adjustable range.

EXAMPLE

The following sets the full-scale range of Channel 1 to 50 mV:

C1:VRNG 50MV

RELATED COMMANDS

GAIN_MODE, VOLT_DIV

STATUS *WAI

Command

DESCRIPTION The *WAI (WAIt to continue) command, required by the IEEE 488.2 standard, has no effect on the instrument, as the LSA1000 only starts processing a command when the previous command has been entirely executed.

COMMAND SYNTAX `*WAI`

RELATED COMMANDS *OPC

ACQUISITION — WAIT Command

DESCRIPTION

The WAIT command prevents the instrument from analyzing new commands until the LSA1000 has completed the current acquisition.

COMMAND SYNTAX

WAIT

EXAMPLE

```
send: "TRMD SINGLE"
loop {send:"ARM;WAIT;C1:PAVA?MAX"
        read response
        process response
        }
```

This example finds the maximum amplitudes of several signals acquired one after another. ARM starts a new data acquisition. The WAIT command ensures that the maximum is evaluated for the newly acquired waveform.

"**C1:PAVA?MAX**" instructs the instrument to evaluate the maximum data value in the Channel 1 waveform.

RELATED COMMANDS

*TRG

WAVEFORM TRANSFER — WAVEFORM, WF

Command/Query

DESCRIPTION

A WAVEFORM command transfers a waveform from the controller to the LSA1000, whereas a WAVEFORM? query transfers a waveform from the LSA1000 to the controller.

The WAVEFORM command stores an external waveform back into the LSA1000's internal memory. A waveform consists of several distinct entities:

1. the descriptor (DESC)
2. the user text (TEXT)
3. the time (TIME) descriptor
4. the data (DAT1) block, and, optionally
5. a second block of data (DAT2).

For further information on the structure of the waveform refer to Chapter 4.

Note 1: Only complete waveforms queried with "WAVEFORM? ALL" can be restored into the LSA1000.

The WAVEFORM? query instructs the LSA1000 to transmit a waveform to the controller. The entities may be queried independently. If the "ALL" parameter is specified, all four or five entities are transmitted in one block in the order enumerated above.

Note 2: The format of the waveform data depends on the current settings specified by the last WAVEFORM_SETUP command, the last COMM_ORDER command, and the last COMM_FORMAT command.

COMMAND SYNTAX

<memory> : **WaveForm ALL** <waveform_data_block>

<memory> : = {**M1**, **M2**, **M3**, **M4**}

<waveform_data_block> : = Arbitrary data block (*see Chapter 5*).

QUERY SYNTAX

<trace> : **WaveForm?** <block>

<trace> : = {**TA**, **TB**, **TC**,**TD**, **M1**, **M2**, **M3**, **M4**, **C1**, **C2**}

<block> : = {**DESC**, **TEXT**, **TIME**, **DAT1**, **DAT2**, **ALL**}

Note 3: If no parameter is given ALL will be assumed.

RESPONSE FORMAT <trace> : **`WaveForm`** <block>**,**<waveform_data_block>

Note 4: It may be convenient to disable the response header if the waveform is to be restored. Refer to command COMM_HEADER for further details.

EXAMPLE The following reads the block DAT1 from Memory 1:

```
M1:WF? DAT1
```

RELATED COMMANDS INSPECT, COMM_FORMAT, COMM_ORDER, FUNCTION_STATE, TEMPLATE, WAVEFORM_SETUP, WAVEFORM_TEXT

WAVEFORM TRANSFER WAVEFORM_SETUP, WFSU

Command/Query

DESCRIPTION

The WAVEFORM_SETUP command specifies the amount of data in a waveform to be transmitted to the controller. The command controls the settings of the parameters listed below.

Notation			
FP	first point	**NP**	number of points
SP	sparsing		

Sparsing (SP): The sparsing parameter defines the interval between data points. For example:

SP = 0	sends all data points
SP = 1	sends all data points
SP = 4	sends every 4th data point

Number of points (NP): The number of points parameter indicates how many points should be transmitted. For example:

NP = 0	sends all data points
NP = 1	sends 1 data point
NP = 50	sends a maximum of 50 data points
NP = 1001	sends a maximum of 1001 data points

First point (FP): The first point parameter specifies the address of the first data point to be sent. For example:

FP = 0	corresponds to the first data point
FP = 1	corresponds to the second data point
FP = 5000	corresponds to data point 5001

The WAVEFORM_SETUP? query returns the transfer parameters currently in use.

COMMAND SYNTAX

WaveForm_SetUp SP,<sparsing>**,NP,**<number>**,FP,**<point>

Note 1: After power-on, all values are set to 0 (i.e. entire waveforms will be transmitted without sparsing).

Note 2: Parameters are grouped in pairs. The first of the pair names the variable to be modified, whilst the second gives the new value to be assigned. Pairs may be given in any order and may be restricted to those variables to be changed.

QUERY SYNTAX	**`WaveForm_SetUp?`**
RESPONSE FORMAT	**`WaveForm_SetUp SP,`** <sparsing>**`,NP`**,<number>,FP,<point>
EXAMPLE	The following command specifies that every 3rd data point (SP=3) starting at address 200 should be transferred: **`WFSU SP,3,FP,200`**
RELATED COMMANDS	INSPECT, WAVEFORM, TEMPLATE

WAVEFORM TRANSFER — WAVEFORM_TEXT, WFTX

Command/Query

DESCRIPTION

The WAVEFORM_TEXT command is used to document the conditions under which a waveform has been acquired. The text buffer is limited to 160 characters.

The WAVEFORM_TEXT? query returns the text section of the specified trace.

COMMAND SYNTAX

<trace> : **WaveForm_TeXt ‘**<text>**’**

<trace> : = {**TA**, **TB**, **TC**, **TD**, **M1**, **M2**, **M3**, **M4**, **C1**, **C2**}
<text> : = An ASCII message (max. 160 characters long)

QUERY SYNTAX

<trace> : **WaveForm_TeXt?**

RESPONSE FORMAT

<trace> : **WaveForm_TeXt “**<text>**”**

EXAMPLE

The following documents Trace A (TA):

```
TA:WFTX ‘Averaged pressure signal. Experiment
      carried out Jan.15, 98’
```

RELATED COMMANDS

INSPECT, WAVEFORM, TEMPLATE

DISPLAY — XY_ASSIGN?, XYAS?

Query

DESCRIPTION

The XY_ASSIGN? query returns the traces currently assigned to the XY display. If there is no trace assigned to the X-axis and/or the Y-axis the value UNDEF will be returned instead of the trace name.

This command is included for use with programs such as ScopeExplorer.

QUERY SYNTAX

XY_ASsign?

RESPONSE FORMAT

XY_ASsign <X_source**>**,<Y_source>

<X_source> : = {**UNDEF**, **TA**, **TB**, **TC**, **TD**, **C1**, **C2**}

<Y_source> : = {**UNDEF**, **TA**, **TB**, **TC**, **TD**, **C1**, **C2**}

EXAMPLE

The following query finds the traces assigned to the X-axis and the Y-axis respectively:

XYAS?

Example of response message:

XYAS C1,C2

RELATED COMMANDS

TRACE

CURSOR — XY_CURSOR_ORIGIN, XYCO

Command/Query

DESCRIPTION

The XY_CURSOR_ORIGIN command sets the position of the origin for XY absolute cursor measurements.

Absolute 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 cursor measurements.

COMMAND SYNTAX

XY_Cursor_Origin <mode>

<mode> : = {**ON**, **OFF**}

QUERY SYNTAX

XY_Cursor_Origin?

RESPONSE FORMAT

XY_Cursor_Origin <mode>

EXAMPLE

The following command sets the origin for absolute cursor measurements to the center of the XY grid.

```
XYCO ON
```

RELATED COMMANDS

XY_CURSOR_VALUE

CURSOR XY_CURSOR_SET, XYCS

Command/Query

DESCRIPTION

The XY_CURSOR_SET command allows the user to position any one of the six independent XY voltage cursors at a given 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).

The CURSOR_SET command is used to position the time cursors.

Notation	
XABS	vertical absolute on X axis
XREF	vertical reference on X axis
XDIF	vertical difference on X axis
YABS	vertical absolute on Y axis
YREF	vertical reference on Y axis
YDIF	vertical difference on Y axis

COMMAND SYNTAX

XY_Cursor_Set
<cursor>,<position>[,<cursor>,<position>,...
<cursor>,<position>]

<cursor> : = {**XABS**, **XREF**, **XDIF**, **YABS**, **YREF**, **YDIF**}

<position> : = –4 to 4 DIV

Note 1: The suffix DIV is optional.

Note 2: Parameters are grouped in pairs. The first of the pair names the cursor to be modified, whilst the second indicates its new value. Pairs may be given in any order and may be restricted to those items to be changed.

QUERY SYNTAX

XY_Cursor_Set? [<cursor,...<cursor>]

<cursor> : = {**XABS**, **XREF**, **XDIF**, **YABS**, **YREF**, **YDIF**, **ALL**}

Note: If <cursor> is not specified, ALL will be assumed.

RESPONSE FORMAT

`XY_Cursor_Set`
<cursor>`,`<position>[`,`<cursor>`,`<position>...`,`
<cursor>`,`<position>]

EXAMPLE

The following command positions the XREF and YDIF at +3 DIV and –2 DIV respectively.

```
XYCS XREF,3DIV,YDIF,-2DIV
```

RELATED COMMANDS

XY_CURSOR_VALUE, CURSOR_MEASURE, CURSOR_SET

CURSOR **XY_CURSOR_VALUE?, XYCV?**
Query

DESCRIPTION

The XY_CURSOR_VALUE? query returns the current values of the X versus Y cursors. The X versus Y trace does not need to be displayed to obtain these parameters, but valid sources must be assigned to the X and Y axes.

Notation	
<cursor type> : = [**HABS**, **HREL**, **VABS**, **VREL**]	
<cursor type>**_X**	X
<cursor type>**_Y**	Y
<cursor type>**_RATIO**	+ Y/ + X
<cursor type>**_PROD**	+ Y* + X
<cursor type>**_ANGLE**	arc tan(+ Y/ + X)
<cursor type>**_RADIUS**	sqrt(+ X* + X + + Y* + Y)

QUERY SYNTAX

XY_Cursor_Value? [<parameter>,...<parameter>]

<parameter> : = {**HABS_X**, **HABS_Y**, **HABS_RATIO**, **HABS_PROD**, **HABS_ANGLE**, **HABS_RADIUS**, **HREL_X**, **HREL_Y**, **HREL_RATIO**, **HREL_PROD**, **HREL_ANGLE**, **HREL_RADIUS**, **VABS_X**, **VABS_Y**, **VABS_RATIO**, **VABS_PROD**, **VABS_ANGLE**, **VABS_RADIUS**, **VREL_X**, **VREL_Y**, **VREL_RATIO**, **VREL_PROD**, **VREL_ANGLE**, **VREL_RADIUS**, **ALL**}

Note: If <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. If no trace has been assigned to either the X axis or the Y axis, an environment error will be generated.

RESPONSE FORMAT

XY_Cursor_Value <parameter>**,**<value>[**,**...<parameter>**,**<value>]

<value> : = A decimal value or UNDEF

EXAMPLE

The following query reads the ratio of the absolute horizontal cursor, the angle of the relative horizontal cursor, and the product of the absolute vertical cursors:

```
XYCV? HABS_RATIO,HREL_ANGLE,VABS_PROD
```

RELATED COMMANDS

CURSOR_MEASURE, CURSOR_VALUE,
XY_CURSOR_ORIGIN

DISPLAY XY_DISPLAY, XYDS

Command/Query

DESCRIPTION

The XY_DISPLAY command enables or disables the XY display mode. When off, the scope is in standard display mode.

The XY_DISPLAY? query returns the current mode of the XY display.

COMMAND SYNTAX

`XY_DiSplay` <mode>

QUERY SYNTAX

`XY_DiSplay?`

RESPONSE FORMAT

`XY_DiSplay` <mode>

EXAMPLE

The following turns the XY display ON:

```
XYDS ON
```

RELATED COMMANDS

GRID

DISPLAY XY_SATURATION, XYSA
Command/Query

DESCRIPTION

The XY_SATURATION command sets the level at which the color spectrum of the persistence display is saturated in XY display mode. The level is specified in terms of percentage (PCT) of the total persistence data map population. A level of 100 PCT corresponds to the color spectrum being spread across the entire depth of the persistence data map. At lower values, the spectrum will saturate (brightest value) at the specified percentage value. The PCT is optional.

The response to the XY_SAT? query indicates the saturation level of the persistence data maps.

COMMAND SYNTAX

`XY_SAturation` <trace>,<value> [<trace>,<value>]

<trace> : = { **`C1`**, **`C2`**, **`TA`**, **`TB`**, **`TC`**, **`TD`**, **`ALL`**}

<value> : = 0 to 100 PCT

Note: The suffix PCT is optional.

QUERY SYNTAX

`XY_SAturation?`

RESPONSE FORMAT

`XY_SAturation` <trace>,<value>

EXAMPLE

The following sets the saturation level of the XY persistence data map for channel 3 to be 60%, i.e. 60% of the data points will be displayed with the color spectrum, with the remaining 40% saturated in the brightest color:

```
XYSA C3,60
```

RELATED COMMANDS

PERSIST_SAT

Waveform Template

This template is the instrument's response to a command of the form "TMPL?":

```
/00
000000              LECROY_2_2:  TEMPLATE
                    8 66 111
;
; Explanation of the formats of waveforms and their descriptors on the
; LeCroy Digital Oscilloscopes,
;     Software Release 44.1.1.1, 94/04/18.
;
; A descriptor and/or a waveform consists of one or several logical data blocks
; whose formats are explained below.
; Usually, complete waveforms are read: at the minimum they consist of
;       the basic descriptor block WAVEDESC
;       a data array block.
; Some more complex waveforms, e.g. Extrema data or the results of a Fourier
; transform, may contain several data array blocks.
; When there are more blocks, they are in the following sequence:
;       the basic descriptor block WAVEDESC
;       the history text descriptor block USERTEXT (may or may not be present)
;       the time array block (for RIS and sequence acquisitions only)
;       data array block
;       auxiliary or second data array block
;
; 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
;                                   with the format shown below
;                                   31  30 .. 23   22 ... 0   bit position
;                                   s   exponent   fraction
;                                   where
;                                   s = sign of the fraction
;                                   exponent = 8 bit exponent e
;                                   fraction = 23 bit fraction f
;                                   and the final value is
;                                   (-1)**s * 2**(e-127) * 1.f
;                     double        64-bit IEEE floating point value
;                                   with the format shown below
;                                   63  62 .. 52   51 ... 0   bit position
;                                   s   exponent   fraction
;                                   where
;                                   s = sign of the fraction
;                                   exponent = 11 bit exponent e
;                                   fraction = 52 bit fraction f
;                                   and the final value is
;                                   (-1)**s * 2**(e-1023) * 1.f
;                       enum        enumerated value in the range 0 to N
;                                   represented as a 16-bit data value.
;                                   The list of values follows immediately.
;                                   The integer is preceded by an _.
;                 time_stamp        double precision floating point number,
;                                   for the number of seconds and some bytes
;                                   for minutes, hours, days, months and year.
;
;                                   double  seconds     (0 to 59)
;                                   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, word or float, depending on the
;                                   read-out mode reflected by the WAVEDESC
;                                   variable COMM_TYPE, modifiable via the
;                                   remote command COMM_FORMAT.
;                       text        arbitrary length text string
;                                   (maximum 160)
;            unit_definition        a unit definition consists of a 48 character
;                                   ASCII string terminated with a null byte
;                                   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          ; chosen by remote command COMM_FORMAT
               _0      byte
               _1      word
               endenum
;
< 34>          COMM_ORDER: enum
               _0      HIFIRST
               _1      LOFIRST
               endenum
;
;
; The following variables of this basic wave descriptor block 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, this
; 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
< 44>          RES_DESC1: long          ;
;
;ARRAYS :
;
< 48>          TRIGTIME_ARRAY: long     ; length in bytes of TRIGTIME array
;
< 52>          RIS_TIME_ARRAY: long     ; length in bytes of RIS_TIME array
;
< 56>          RES_ARRAY1: long         ; an expansion entry is reserved
;
< 60>          WAVE_ARRAY_1: long       ; length in bytes of 1st simple
                                        ; data array. In transmitted waveform,
                                        ; represent the number of transmitted
                                        ; bytes in accordance with the NP
                                        ; parameter of the WFSU remote command
                                        ; and the used format (see COMM_TYPE).
;
< 64>          WAVE_ARRAY_2: long       ; length in bytes of 2nd simple
                                        ; data array
```

```
;
< 68>          RES_ARRAY2: long
< 72>          RES_ARRAY3: long        ; 2 expansion entries are reserved
;
; The following variables identify the instrument
;
< 76>          INSTRUMENT_NAME: string
;
< 92>          INSTRUMENT_NUMBER: long
;
< 96>          TRACE_LABEL: string     ; identifies the waveform.
;
<112>          RESERVED1: word
<114>          RESERVED2: word         ; 2 expansion entries
;
; The following variables describe the waveform and the time at
; which the waveform was generated.
;
<116>          WAVE_ARRAY_COUNT: long  ; number of data points in the data
                                       ; array. If there are two data
                                       ; arrays (FFT or Extrema), this number
                                       ; applies to each array separately.
;
<120>          PNTS_PER_SCREEN: long   ; nominal number of data points
                                       ; on the screen
;
<124>          FIRST_VALID_PNT: long   ; count of number of points to skip
                                       ; before first good point
                                       ; FIRST_VALID_POINT = 0
                                       ; for normal waveforms.
;
<128>          LAST_VALID_PNT: long    ; index of last good data point
                                       ; in record before padding (blanking)
                                       ; was started.
                                       ; LAST_VALID_POINT = WAVE_ARRAY_COUNT-1
                                       ; except for aborted sequence
                                       ; and rollmode acquisitions
;
<132>          FIRST_POINT: long       ; for input and output, indicates
                                       ; the offset relative to the
                                       ; beginning of the trace buffer.
                                       ; Value is the same as the FP parameter
                                       ; of the WFSU remote command.
;
<136>          SPARSING_FACTOR: long   ; for input and output, indicates
                                       ; the sparsing into the transmitted
                                       ; data block.
                                       ; Value is the same as the SP parameter
```

```
                                                 ; of the WFSU remote command.
;
<140>          SEGMENT_INDEX: long       ; for input and output, indicates the
                                         ; index of the transmitted segment.
                                         ; Value is the same as the SN parameter
                                         ; of the WFSU remote command.
;
<144>          SUBARRAY_COUNT: long      ; for Sequence, acquired segment count,
                                         ; between 0 and NOM_SUBARRAY_COUNT
;
<148>          SWEEPS_PER_ACQ: long      ; for Average or Extrema,
                                         ; number of sweeps accumulated
                                         ; else 1
;
<152>          POINTS_PER_PAIR: word     ; for Peak Dectect waveforms (which always
                                         ; include data points in DATA_ARRAY_1 and
                                         ; min/max pairs in DATA_ARRAY_2).
                                         ; Value is the number of data points for
                                         ; each min/max pair.
;
<154>          PAIR_OFFSET: word         ; for Peak Dectect waveforms only
                                         ; Value is the number of data points by
                                         ; which the first min/max pair in
                                         ; DATA_ARRAY_2 is offset relative to the
                                         ; first data value in DATA_ARRAY_1.
;
<156>          VERTICAL_GAIN: float
;
<160>          VERTICAL_OFFSET: float    ; to get floating values from raw data :
                                         ; VERTICAL_GAIN * data - VERTICAL_OFFSET
;
<164>          MAX_VALUE: float          ; maximum allowed value. It corresponds
                                         ; to the upper edge of the grid.
;
<168>          MIN_VALUE: float          ; minimum allowed value. It corresponds
                                         ; to the lower edge of the grid.
;
<172>          NOMINAL_BITS: word        ; a measure of the intrinsic precision
                                         ; of the observation: ADC data is 8 bit
                                         ;    averaged data is 10-12 bit, etc.
;
<174>          NOM_SUBARRAY_COUNT: word ; for Sequence, nominal segment count
                                         ; else 1
;
<176>          HORIZ_INTERVAL: float     ; sampling interval for time domain
                                         ;  waveforms
;
<180>          HORIZ_OFFSET: double      ; trigger offset for the first sweep of
                                         ; the trigger, seconds between the
```

```
                                        ; trigger and the first data point
;
<188>          PIXEL_OFFSET: double     ; needed to know how to display the
                                        ; waveform
;
<196>          VERTUNIT: unit_definition  ; units of the vertical axis
;
<244>          HORUNIT: unit_definition   ; units of the horizontal axis
;
<292>          RESERVED3: word
<294>          RESERVED4: word          ; 2 expansion entries
;
<296>          TRIGGER_TIME: time_stamp ; time of the trigger
;
<312>          ACQ_DURATION: float      ; duration of the acquisition (in sec)
                                        ; in multi-trigger waveforms.
                                        ; (e.g. sequence, RIS,  or averaging)
;
<316>          RECORD_TYPE: enum
               _0      single_sweep
               _1      interleaved
               _2      histogram
               _3      graph
               _4      filter_coefficient
               _5      complex
               _6      extrema
               _7      sequence_obsolete
               _8      centered_RIS
               _9      peak_detect
               endenum
;
<318>          PROCESSING_DONE: enum
               _0       no_processing
               _1       fir_filter
               _2       interpolated
               _3       sparsed
               _4       autoscaled
               _5       no_result
               _6       rolling
               _7       cumulative
               endenum
;
<320>          RESERVED5: word          ; expansion entry
;
<322>          RIS_SWEEPS: word         ; for RIS, the number of sweeps
                                        ; else 1
;
; The following variables describe the basic acquisition
; conditions used when the waveform was acquired
;
<324>          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
               _100  EXTERNAL
               endenum
;
<326>          VERT_COUPLING: enum
               _0      DC_50_Ohms
               _1      ground
               _2      DC_1MOhm
               _3      ground
               _4      AC,_1MOhm
               endenum
;
<328>          PROBE_ATT: float
;
<332>          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
;
<334>          BANDWIDTH_LIMIT: enum
               _0      off
               _1      on
               endenum
;
<336>          VERTICAL_VERNIER: float
;
<340>          ACQ_VERT_OFFSET: float
;
<344>          WAVE_SOURCE: enum
               _0      CHANNEL_1
               _1      CHANNEL_2
               _2      CHANNEL_3
               _3      CHANNEL_4
               _9      UNKNOWN
               endenum
;
/00              ENDBLOCK
;
;===========================================================================
;
USERTEXT: BLOCK
;
; Explanation of the descriptor block USERTEXT at most 160 bytes long.
;
;
<  0>          TEXT: text               ; a list of ASCII characters
;
/00              ENDBLOCK
;
;===========================================================================
;
DATA_ARRAY_1: ARRAY
;
; Explanation of the data array DATA_ARRAY_1.
; This main data array is always present. It is the only data array for
; most waveforms.
; The data item is repeated for each acquired or computed data point
; of the first data array of any waveform.
;
<  0>          MEASUREMENT: data        ; the actual format of a data is
                                        ; given in the WAVEDESC descriptor
                                        ; by the COMM_TYPE variable.
;
/00              ENDARRAY
;
;===========================================================================
;
DATA_ARRAY_2: ARRAY
;
; Explanation of the data array DATA_ARRAY_2.
; This is an optional secondary data array for special types of waveforms:
;       Complex FFT     imaginary part      (real part in DATA_ARRAY_1)
;       Extrema         floor trace         (roof trace in DATA_ARRAY_1)
;       Peak Detect     min/max pairs       (data values in DATA_ARRAY_1)
; In the first 2 cases, there is exactly one data item in DATA_ARRAY_2 for
; each data item in DATA_ARRAY_1.
; In Peak Detect waveforms, there may be fewer data values in DATA_ARRAY_2,
; as described by the variable POINTS_PER_PAIR.
;
<  0>          MEASUREMENT: data        ; the actual format of a data is
                                        ; given in the WAVEDESC descriptor
                                        ; by the COMM_TYPE variable.
;
/00              ENDARRAY
;
;==========================================================================
;
TRIGTIME: ARRAY
;
; Explanation of the trigger time array TRIGTIME.
; This optional time array is only present with SEQNCE waveforms.
; The following data block is repeated for each segment which makes up
; the acquired sequence record.
;
<  0>          TRIGGER_TIME: double     ; for sequence acquisitions,
                                        ; time in seconds from first
                                        ; trigger to this one
;
<  8>          TRIGGER_OFFSET: double   ; the trigger offset is in seconds
                                        ; from trigger to zeroth data point
;
/00              ENDARRAY
;
;==========================================================================
;
RISTIME: ARRAY
;
; Explanation of the random-interleaved-sampling (RIS) time array RISTIME.
; This optional time array is only present with RIS waveforms.
; This data block is repeated for each sweep which makes up the RIS record
;
<  0>          RIS_OFFSET: double       ; seconds from trigger to zeroth
                                        ; point of segment
;
/00              ENDARRAY
;
;==========================================================================
;
```

```
SIMPLE: ARRAY
;
; Explanation of the data array SIMPLE.
; This data array is identical to DATA_ARRAY_1. SIMPLE is an accepted
; alias name for DATA_ARRAY_1.
;
<  0>          MEASUREMENT: data        ; the actual format of a data is
                                        ; given in the WAVEDESC descriptor
                                        ; by the COMM_TYPE variable.
;
/00               ENDARRAY
;
;===========================================================================
;
DUAL: ARRAY
;
; Explanation of the DUAL array.
; This data array is identical to DATA_ARRAY_1, followed by DATA_ARRAY_2.
; DUAL is an accepted alias name for the combined arrays DATA_ARRAY_1 and
; DATA_ARRAY_2 (e.g. real and imaginary parts of an FFT).
;
<  0>          MEASUREMENT_1: data      ; data in DATA_ARRAY_1.
;
<  0>          MEASUREMENT_2: data      ; data in DATA_ARRAY_2.
;
/00               ENDARRAY
;
;
000000                ENDTEMPLATE
```

A Template

Index