HEWLETT PACKARD

HP 35660A Dynamic Signal Analyzer
HP-IB Programming Reference

Manual Part No. 35660-90025
Microfiche Part No. 35660-90225

8600 Soper Hill Road
Everett, Washington 98205-1298 U.S.A.

Printed: July 1988

NOTICE

The information contained in this document is subject to change without notice.

HEWLETT-PACKARD MAKES NO WARRANTY OF ANY KIND WITH REGARD TO THIS MANUAL, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE. Hewlett-Packard shall not be liable for errors contained herein or direct, indirect, special, incidental or consequential damages in connection with the furnishing, performance, or use of this material.

WARRANTY

A copy of the specific warranty terms applicable to your Hewlett-Packard product and replacement parts can be obtained from your local Sales and Service Office.

Use of this manual and flexible disc(s) or tape cartridge(s) supplied for this pack is restricted to this product only. Additional copies of the programs can be made for security and back-up purposes only.

RESTRICTED RIGHTS LEGEND

Use, duplication, or disclosure by the government is subject to restrictions as set forth in subdivision (c) (1) (ii) of the Rights in Technical Data and Computer Software clause at 52.227-7013.

HEWLETT-PACKARD COMPANY

3000 Hanover St.

Palo Alto, CA 94303

Table of Contents

Chapter 1
Introduction to HP-IB

Notice to Experienced HP-IB Programmers

Two things that have been true about HP-IB programming for many past instruments are not true for HP-IB programming of the HP 35660A.

In past instruments, a command typically consisted of a single mnemonic. An HP 35660A command typically consists of a series of mnemonics that are selected from a command hierarchy. The hierarchy organizes commands into groups that access related analyzer functions. These multi-mnemonic commands are less cryptic than single-mnemonic commands and help to make your programs more self-documenting. Chapter 3, describes the command hierarchy.

It has also been typical for past instruments to have an HP-IB command for each front-panel hardkey and softkey. This is not true for the HP 35660A. The analyzer does give you HP-IB access to all front-panel functions, but there is not a one-to-one correspondence between commands and keys. This results from the fact that the HP-IB command hierarchy is organized differently than the front-panel key hierarchy. Appendix B provides a cross-reference for selecting an HP-IB command that is equivalent to a series of front-panel key presses. A special front-panel feature called Mnemonic Echo also provides cross-referencing. Mnemonic Echo is described in Appendix B.

Manual Overview

This manual is organized into five major parts:

1. Programming Fundamentals — This part of the manual contains five chapters, each of which discusses some aspect of programming the HP 35660A via the HP-IB:
   - Chapter 1 introduces you to HP-IB concepts and tells you how to configure the HP 35660A in an HP-IB system.
   - Chapter 2 explains how the analyzer interacts with the controller and other devices on the HP-IB.
   - Chapter 3 explains the HP 35660A's command hierarchy.
   - Chapter 4 explains how data is transferred between the analyzer and a controller. It also describes the structure of files you can save from the analyzer.
   - Chapter 5 describes the analyzer's register structure and tells you how the analyzer uses registers to generate service requests.
2. Programming Examples — This part (Chapter 6) contains commented programming examples.
3. Command Reference — This part (Chapter 7) contains a detailed description of each HP-IB command. The command descriptions are organized alphabetically.
4. Appendices — This part of the manual contains four appendices:
   - Appendix A explains how to select the vertical units you can send with certain commands.
   - Appendix B cross-references front-panel keys to equivalent HP-IB commands.
   - Appendix C lists all of the HP-IB commands in alphabetical order.
   - Appendix D lists the analyzer's error messages.
5. Index — This part of the manual references the page numbers where different subjects are discussed. It can be especially useful for determining which command you should use to access a particular analyzer function.

HP-IB Overview

HP-IB, the Hewlett-Packard Interface Bus, is a high performance bus that allows you to build integrated test systems from individual instruments and computers. The bus and its associated interface operations are defined by the IEEE 488.1 standard.

HP-IB cables provide the physical link between devices on the bus. There are eight data lines on each cable that are used to send data from one device to another. Devices that can be addressed to send data over these lines are called "talkers," and those that can be addressed to receive data are called "listeners." There are also five control lines on each cable that are used to manage traffic on the data lines and to control other interface operations. Devices that can use these control lines to specify the talker and listener in a data exchange are called "controllers."

When an HP-IB system contains more than one device with controller capabilities, only one of the devices is allowed to control data exchanges at any given time. The device currently controlling data exchanges is called the "active controller." Also, only one of the controller-capable devices can be designated as the "system controller." The system controller is the one device that can take control of the bus even if it is not the active controller. The HP 35660A can act as a talker, listener, active controller, or system controller at different times.

HP-IB addresses provide a way to identify devices on the bus. For example, the active controller uses HP-IB addresses to specify which device talks and which device listens during a data exchange. This means that each device's address must be unique. You set a device's address on the device itself, usually using a rear-panel switch or a front-panel key sequence.

Sending Commands Over the HP-IB

Commands are sent over the HP-IB via a controller's language system, such as BASIC or Pascal. As a result, you will need to determine which keywords your controller's language system uses to send HP-IB commands. When looking for keywords, keep in mind that there are actually two different kinds of HP-IB commands:

- Bus management commands, which control the HP-IB interface
- Device commands, which control analyzer functions

Language systems usually deal differently with these two kinds of HP-IB commands. For example, HP BASIC 5.0 uses a unique keyword to send each bus management command, but always uses the keyword OUTPUT to send device commands. For more information on the differences between bus management commands and device commands, see Chapter 2, "Behavior in an HP-IB System."

The following example shows how to send a typical device command:

```
OUTPUT 711;"AVERAGE:COUNT 5"
```

This sends the command within the quotes (AVERAGE:COUNT 5) to the HP-IB device at address 711. If the device is an HP 35660A, the command instructs the analyzer to set the number of averages to 5.

NOTE All examples in this manual are written for HP BASIC 5.0 running on an HP Series 200 computer.

The IEEE 488.1 and 488.2 Standards

The HP 35660A conforms to both the IEEE 488.1 standard and the IEEE 488.2 standard. The IEEE 488.1 standard defines the mechanical, electrical, and functional aspects of the original 488 bus. The IEEE 488.2 standard defines data encoding, data formats, bus communication protocols, and a set of commonly needed commands for instruments that use the 488 bus.

HP-IB Setup

This section contains a procedure for configuring the HP 35660A and an external controller in a simple HP-IB system. Although an HP Series 200 computer is the controller used in the system, other computers that support an HP-IB interface can also be used. If you are using one of those other computers, the configuration procedure can only be used as a general guide. You should consult your computer's documentation for more complete information.

This section also contains a procedure for verifying that commands can be sent over the HP-IB. HP BASIC is used for the verification procedure's test program. If your computer uses some other language, the keywords and syntax for the test program may be different. If this is the case, you will need to write a similar program using your language's keywords and syntax.

Configuring the HP-IB System

Equipment and Software

HP 35660A Dynamic Signal Analyzer
HP 9836 computer
HP 10833A, B, C, or D HP-IB Cable
HP BASIC 5.0

Procedure

1. Turn off the HP 35660A and the HP 9836, then connect them with the HP-IB cable as shown in Figure 1-1.

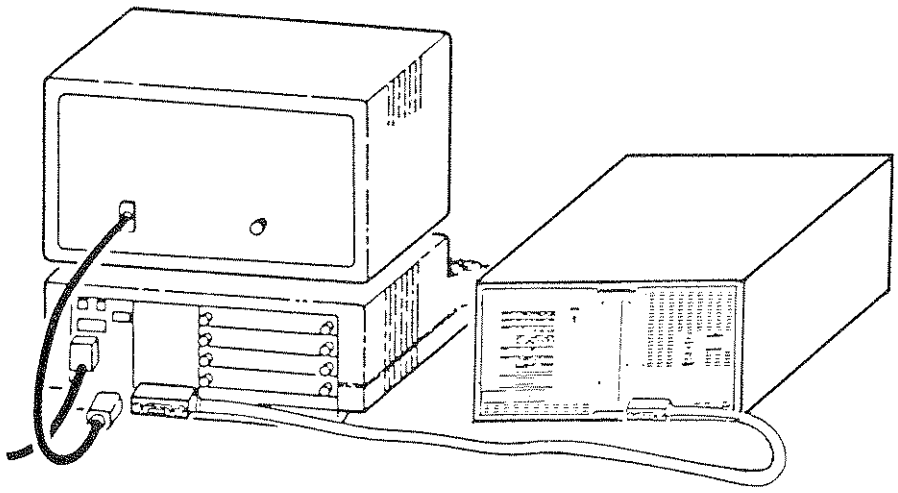


Figure 1-1. HP-IB Connections

2. Turn on the HP 9836. If necessary, load HP BASIC 5.0 following the instructions in the computer's operating manual. Note that the following language extensions must be installed for the verification program to work:
   - CRTA
   - HPIB
   - IO
   - EDIT

   Programs that are more complex than the verification program will probably require more language extensions.

3. Turn on the HP 35660A. When the softkey labels appear, press the Local/HP-IB hardkey. (see Figure 1-2)

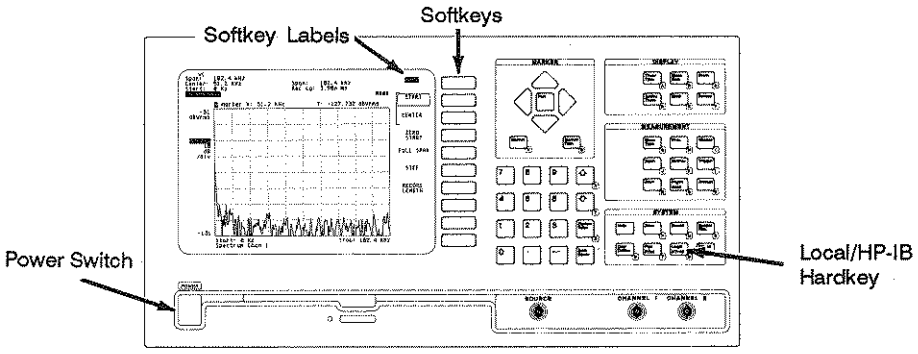


Figure 1-2. HP 35660A Front Panel

4. Verify that the analyzer's address is set to 11. The current address setting is displayed in the HpibAddr field (see Figure 1-3). You can change the analyzer's address by pressing the ANALYZER ADDRESS softkey, then using the numeric keypad and the ENTER softkey to enter a new value. However, the instructions in the verification procedure assume that the analyzer address is set to 11.

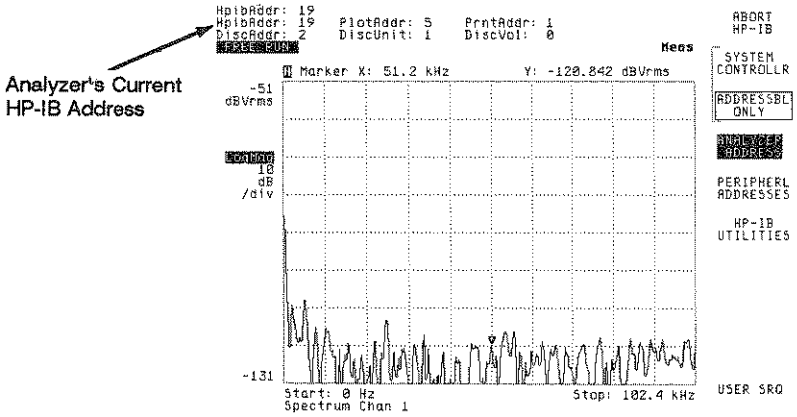


Figure 1-3. HP 35660A Screen After Pressing Local/HP-IB

5. Verify that the analyzer is set to the addressable-only mode. The softkey labels that appear when you press the Local/HP-IB hardkey include SYSTEM CONTROLLR and ADDRESSBL ONLY. Only one of these two softkeys can be selected at a time, and the one that is selected will have a box around it. If ADDRESSBL ONLY is not selected, then press that softkey.

NOTE In any HP-IB system there can be more than one device with controller capabilities. But at any given time, only one device on the bus can be designated as the system controller.

Quick Verification

Having just completed all the steps in the preceding section, you are ready to verify that commands can be sent over the HP-IB. In this quick verification, you are going to enter an HP BASIC keyword that should place the HP 35660A under remote control.

Procedure

1. Press the Local/HP-IB hardkey, then the HP-IB UTILITIES softkey. Another set of softkey labels is displayed. In the first label, STATUS ON/OFF, OFF should be highlighted.
2. Press the STATUS ON/OFF softkey so that ON is highlighted. This will display the four HP-IB status indicators: Rmt, Tlk, Ltn, and Srq. (see Figure 1-4)

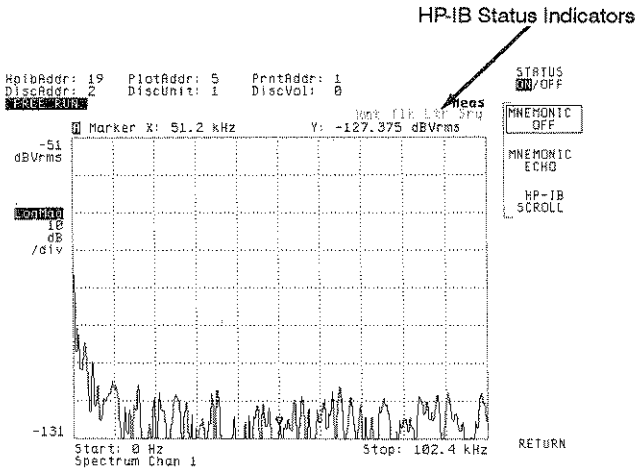


Figure 1-4. HP 35660A Screen with HP-IB Status Indicators

3. Type the following on the computer:

   REMOTE 711

   Then press the computer's ENTER key. Now the HP 35660A's Rmt and Ltn indicators should both be highlighted. This tells you that the analyzer is under remote control of the computer. To return the analyzer to front-panel control, press the Local/HP-IB hardkey or enter the following on the computer:

   LOCAL 711

Troubleshooting

If the Rmt indicator doesn't perform as expected, check the following things:

- Be sure that your HP-IB cable connections are secure and that the cable is free of defects.
- Verify that the analyzer is in addressable-only mode and that its address is set to 11.
- Be sure you are using the required equipment and software.
- Be sure you have loaded all the required language extensions into the computer. (For a list of loaded extensions, enter the following into the computer: LIST BIN)

If everything seems to be in order, but the Rmt indicator still doesn't perform as expected, contact your local HP Sales/Service office.

Verification Program

The quick verification procedure confirmed that the computer could talk to the analyzer. However, you must write a short program to confirm that the analyzer can talk to the computer. If you enter the program correctly, the computer displays the following statement when you run the program:

ACTUAL FREQUENCY SPAN IS: 25600 HZ

NOTE The following procedure assumes that you have completed all the steps in "Configuring the HP-IB System" using all the required equipment and software.

Procedure

1. Enter the following program:

```
10    PRINTER IS 1
20    Dsa=711
30    ABORT 7
40    CLEAR Dsa
50    OUTPUT Dsa;"*RST"
60    OUTPUT Dsa;"FREQ:SPAN 20KHZ"
70    OUTPUT Dsa;"FREQ:SPAN?"
80    ENTER Dsa;A
90    PRINT "ACTUAL FREQUENCY SPAN IS:";A;"HZ"
100   END
```

   See your computer and software documentation if you need help entering the program.

2. Press the computer's RUN key. The program tells the analyzer to preset. It then tells the analyzer to select the nearest frequency span that is greater than or equal to 20 kHz. Finally, the program asks the analyzer to return the selected frequency span and has the computer display the returned value as follows:

   ACTUAL FREQUENCY SPAN IS: 25600 HZ

Troubleshooting

If the program doesn't run correctly, be sure you have entered the program exactly as listed. Then refer to "Quick Verification" for additional troubleshooting hints.

Chapter 2
Behavior in an HP-IB System

HP-IB Interface Capabilities

The HP 35660A has the following interface capabilities, as defined by the IEEE 488.1 standard:

SH1	complete Source handshake capability
AH1	complete Acceptor handshake capability
T6	basic Talker, Serial Poll, no Talk Only, unaddress if MLA
TE0	no Extended Talker capability
L4	basic Listener, no Listen Only, unaddress if MTA
LE0	no Extended Listener capability
SR1	complete Service Request capability
RL1	complete Remote/Local capability
PP0	no Parallel Poll capability
DC1	complete Device Clear capability
DT1	complete Device Trigger capability
C1	System Controller capability
C2	send IFC and take charge Controller capability
C3	send REN Controller capability
C12	send IF messages, receive control, pass control
E2	three-state drivers

Controller Capabilities

The HP 35660A can either be configured as an HP-IB system controller or as an addressable-only HP-IB device. This is done by selecting either the SYSTEM CONTROLLR or ADDRESSBL ONLY softkey on the analyzer's front panel. (These keys are presented when you press the Local/HP-IB hardkey.)

Normally, the HP 35660A is not configured as the system controller unless it is the only controller on the bus. Such a setup would be likely if you just wanted to control printers, plotters, or external disc drives with the analyzer. It might also be the case if you were using HP Instrument BASIC (HP 35680A) to control other test equipment.

When the analyzer is being used with another controller on the bus, it is normally configured as an addressable-only HP-IB device. In this configuration, the analyzer can function as the active controller (when it is passed control), or as a talker or listener.

Bus Management Commands vs. Device Commands

The HP-IB contains an attention (ATN) line that determines when the interface is in the command mode or the data mode. When the interface is in the command mode (ATN TRUE), a controller can send bus management commands over the bus. Bus management commands are used to:

- Specify which devices on the interface can talk (send data) and which can listen (receive data)
- Instruct devices on the bus, either individually or collectively, to perform a particular interface operation

The analyzer's responses to bus management commands are described in the next section.

When the interface is in the data mode, device commands and data can be sent over the bus. Device commands are sent by the controller, but data can be sent either by the controller or a talker. The HP 35660A responds to two different kinds of device commands:

- Common commands, which access device functions required by the IEEE 488.2 standard.
- Device-specific commands, which access the bulk of the analyzer's functions.

The analyzer's responses to device commands are described in Chapter 7 "Command Reference."

Response to Bus Management Commands

This section tells you how the HP 35660A responds to the HP-IB bus management commands. The commands themselves are defined by the IEEE 488.1 standard. Refer to the documentation for your controller's language system to determine how to send these commands.

Device Clear (DCL)

This command causes the analyzer to:

- Clear its input buffer and output queue
- Reset its command parser so it is ready to receive a new program message
- Cancel any pending *OPC command or query

The command does not affect:

- Front-panel operation
- Any analyzer operations in progress (other than those already mentioned)
- Any of the analyzer's settings or registers (although clearing the output queue may indirectly affect the Status Byte's MAV bit)

Go To Local (GTL)

This command returns the analyzer to local (front-panel) control. All keys on the analyzer's front-panel are enabled.

Group Execute Trigger (GET)

This command triggers the analyzer (causes it to start collecting a time record) if the following two things are true:

- The trigger source must be the HP-IB (TRIG:SOUR BUS).
- The analyzer must be ready to trigger. (Bit 2 of the Device Status condition register must be set.)

GET has the same effect as the *TRG and TRIG:IMM program messages.

Interface Clear (IFC)

This command causes the analyzer to halt all bus activity. It discontinues any input or output, although the input buffer and output queue are not cleared. If the analyzer is designated as the active controller when this command is received, it relinquishes control of the bus to the system controller. If the analyzer is enabled to respond to a Serial Poll it becomes Serial Poll disabled.

Local Lockout (LLO)

This command causes the analyzer to enter the local lockout mode, regardless of whether it is in the local or remote mode. The analyzer only leaves the local lockout mode when the HP-IB's Remote Enable (REN) line is set FALSE.

Local lockout ensures that the analyzer's Local/HP-IB key is disabled when the analyzer is in the remote mode. When the key is enabled, it allows a front-panel operator to return the analyzer to local mode, thus enabling all other front-panel keys. However, when the key is disabled, it does not allow the operator to return the analyzer to local mode.

Parallel Poll

The HP 35660A ignores all of the following parallel poll commands:

- Parallel Poll Configure (PPC)
- Parallel Poll Unconfigure (PPU)
- Parallel Poll Enable (PPE)
- Parallel Poll Disable (PPD)

Remote Enable (REN)

REN is a single line on the HP-IB. When it is set TRUE, the analyzer will enter the remote mode when addressed to listen. It will remain in remote mode until it receives the Go to Local (GTL) command or until the REN line is set FALSE.

When the analyzer is in remote mode and local lockout mode, all front-panel keys are disabled. When the analyzer is in remote mode but not in local lockout mode, all front-panel keys are disabled except for the Local/HP-IB key. See Local Lockout for more information.

Selected Device Clear (SDC)

The analyzer responds to this command in the same way that it responds to the Device Clear command. See the latter for details.

Serial Poll

The analyzer responds to both of the serial poll commands. The Serial Poll Enable (SPE) command causes the analyzer to enter the serial poll mode. While the analyzer is in this mode, it sends the contents of its Status Byte register to the controller when addressed to talk.

When the Status Byte is returned in response to a serial poll, bit 6 acts as the Request Service (RQS) bit. If the bit is set, it will be cleared after the Status Byte is returned.

The Serial Poll Disable (SPD) command causes the analyzer to leave the serial poll mode.

Take Control Talker (TCT)

If the analyzer is addressed to talk, this command causes it to take control of the HP-IB. It becomes the active controller on the bus. The analyzer automatically passes control back when it completes the operation that required it to take control. Control is passed back to the address specified by the *PCB program message (which should be sent prior to passing control).

If the analyzer does not require control when this command is received, it immediately passes control back.

Message Exchange

The analyzer communicates with the controller and other devices on the HP-IB via program messages and response messages. Program messages are used to send commands, queries, and data to the analyzer. Response messages are used to return data from the analyzer. The syntax for both kinds of messages is discussed in Chapter 3.

There are two important things to remember about the message exchanges between the analyzer and other devices on the bus:

- The analyzer only talks after it receives a terminated query. (Query termination is discussed in "Query Response Generation," later in this chapter.
- Once it receives a terminated query, the analyzer expects to talk before it is told to do something else.

Buffers and Queues

Buffers and queues enhance the exchange of messages between the HP 35660A and other devices on the bus. The analyzer contains:

- An input buffer
- An error queue
- An output queue

Input Buffer

The input buffer temporarily stores all of the following until they are read by the analyzer's command parser:

- Device commands and queries
- Group Execute Trigger (a bus management command)
- The HP-IB END message (EOI asserted while the last data byte is on the bus)

The input buffer makes it possible for a controller to send multiple program messages to the analyzer without regard to the amount of time required to parse and execute those messages. The buffer holds up to 256 bytes. It is cleared when you do one of the following:

- Turn the analyzer on
- Send Device Clear (DCL) or Selected Device Clear (SDC)
- Press the Local/HP-IB hardkey

Error Queue

The error queue temporarily stores up to ten error messages. Each time the analyzer detects an error, it places a message in the queue. When you send the SYST:ERR query, one message is moved from the error queue to the output queue so it can be read by the controller. Error messages are delivered to the output queue in the order they were received. The error queue is cleared when you do one of the following:

- Turn the analyzer on
- Send the *CLS command

Output Queue

The output queue temporarily stores a single response message until it is read by a controller. The analyzer's output queue holds up to 8192 bytes. It is cleared when you do one of the following:

- Turn the analyzer on
- Send Device Clear (DCL) or Selected Device Clear (SDC)
- Press the Local/HP-IB hardkey

Command Parser

The command parser reads program messages from the input buffer in the order they were received from the bus. It analyzes the syntactic elements of the messages to determine what actions the analyzer should take.

One of the parser's most important functions is to determine a program message's position in the analyzer's command tree. (For more information on the command tree, see Chapter 3.) When the command parser is reset, the next syntactic element it receives is expected to arise from the base of the analyzer's command tree. The parser is reset when you do one of the following:

- Turn the analyzer on
- Send Device Clear (DCL) or Selected Device Clear (SDC)
- Press the Local/HP-IB hardkey
- Follow a semicolon with a colon in a program message (For more information, see "Sending Multiple Commands" in Chapter 3.)

Query Response Generation

When the HP 35660A parses a query, the response to that query is placed in the analyzer's output queue. You should read a query response immediately after sending the query. This ensures that the response will not be cleared before it is read. The response will be cleared before you read it if any of the following message exchange conditions occur:

- Unterminated condition — This condition results when you neglect to properly terminate the query (with an ASCII line feed character or the HP-IB END message) before you read the response.
- Interrupted condition — This condition results when you send a second program message before reading the response to the first.
- Buffer deadlock — This condition results when you send a program message that:
    a. Is longer than the input buffer and
    b. Generates more response data than will fit in the output queue.

Synchronization

This section describes tools you can use to synchronize the analyzer and a controller. Proper use of these tools ensures that the analyzer will be in a known state when you send a particular command or query.

Overlapped Commands

Device commands can be divided into two broad classes:

- Sequential commands
- Overlapped commands

Most device commands that you send to the analyzer are processed sequentially. A sequential command holds off the processing of any subsequent command until it has been completely processed. However, some commands do not hold off the processing of subsequent commands; they are referred to as overlapped commands.

Typically, overlapped commands take longer to process than sequential commands. For example, the INIT:STAT STAR command is used to start a measurement. The command is not considered to have been completely processed until the measurement is complete. This can take a very long time if the measurement is averaging a large number of time records.

NOTE INIT:STAT STAR and INIT:STAT RUN are considered to be pending overlapped commands whenever bit 7 of the Device Status condition register is set to 1. See Chapter 5 for a description of that bit.

The analyzer uses an Operation Complete (OPC) flag to keep track of overlapped commands that are still pending (not completed). The OPC flag is reset to 0 when an overlapped command is pending. It is set to 1 when no overlapped commands are pending. You can not read the OPC flag directly, but all of the following common commands cause the analyzer to take some action based on the setting of the flag:

- *WAI – forces the analyzer to wait until the OPC flag is set to 1, but does not affect the controller
- *OPC? – forces the controller to wait until the OPC flag is set to 1
- *OPC – informs the controller when the OPC flag is set to 1, but leaves it free to perform other tasks until it receives a service request

Each command requires a different amount of overhead in your program. *WAI requires the least overhead, *OPC requires the most.

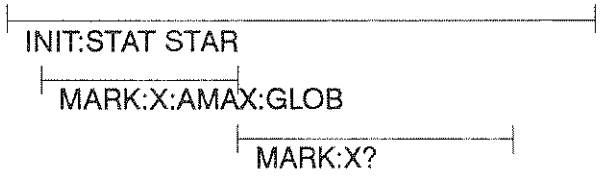
*WAI

This command holds off the processing of subsequent device commands until all overlapped commands are completed (the OPC flag is set to 1). An example will demonstrate the effect of the *WAI command.

Suppose you want to determine which frequency component of a signal contains the greatest amount of energy. You might send the following series of commands:

```
OUTPUT 711;"INIT:STAT STAR"          !Start the measurement.
OUTPUT 711;"MARK:X:AMAX:GLOB"        !Search for max energy.
OUTPUT 711;"MARK:X?"                 !Which frequency?
```

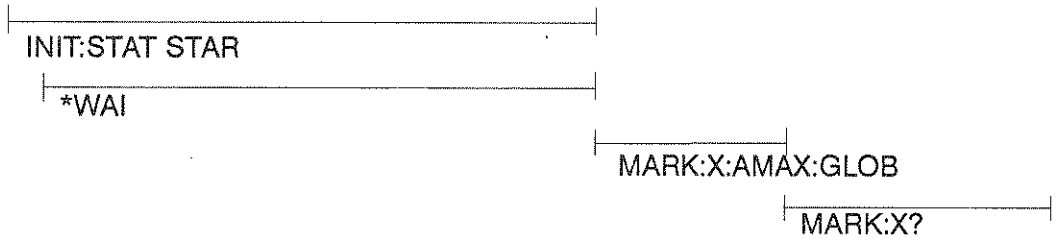
The following timeline shows how the processing times of the three commands relate to each other.



As you can see, INIT:STAT STAR is an overlapped command because it does not hold off the processing of MARK:X:AMAX:GLOB. You may also recall that MARK:X:AMAX:GLOB is not considered complete until the measurement is complete. So in this example, the marker searches for maximum energy before the measurement is complete. This may result in the MARK:X query returning an incorrect value. To solve the problem, you can insert a *WAI command.

```
OUTPUT 711;"INIT:STAT STAR"          !Start the measurement.
OUTPUT 711;"*WAI"                    !Wait until complete.
OUTPUT 711;"MARK:X:AMAX:GLOB"        !Search for max energy.
OUTPUT 711;"MARK:X?"                 !Which frequency?
```

The timeline now looks like this.



The *WAI command keeps the search from taking place until the measurement is completed. The MARK:X query will return the correct value.

*OPC? and *OPC

If you send *OPC?, 1 is placed in the analyzer's output queue when the OPC flag is set to 1. This allows you to effectively pause the controller until all pending overlapped commands are completed. Just design your program so that it must read the queue before it continues.

*OPC? does not hold off the processing of subsequent commands; it only informs you when the OPC flag is set to 1. As a result, you should not send additional overlapped commands to the analyzer between the time you send *OPC? and the time you read 1 from the output queue.

If you send *OPC, bit 0 of the Event Status register is set to 1 when the OPC flag is set to 1. This allows you to use the analyzer's register structure to generate a service request when all pending overlapped commands are completed. However, your program must also have enabled bit 0 of the Event Status register and bit 5 of the Status Byte register. When you synchronize the analyzer and controller in this manner, the controller is free to perform some other task until the service request is generated.

*OPC does not hold off the processing of subsequent commands; it only informs you when the OPC flag is set to 1. As a result, you should not send any commands to the analyzer between the time you send *OPC and the time you receive a service request.

Delayed Result Commands

Delayed result commands change analyzer settings, but the changes they make do not necessarily affect the current measurement. After sending one or more delayed result commands you must always restart your measurement with the INIT:STAT STAR command. This ensures that changes made by delayed result commands will affect the measurement.

Passing Control

The analyzer requires temporary control of the HP-IB to complete some commands. (In the description of each command, a field called "Pass control required" indicates whether or not the command requires control of the bus.) After sending such a command, the active controller must pass control to the analyzer. When the analyzer completes the command, it automatically passes control of the bus back to the controller. For control to be passed back and forth smoothly, you must take steps to ensure that:

- The analyzer has the correct address of the controller so that it can pass control back when the command is completed
- The controller will be informed when control has been passed back

A procedure for passing control follows:

1. Send the controller's HP-IB address with the *PCB command.
2. Clear the analyzer's status registers by sending the *CLS command.
3. Enable the analyzer's status registers to generate a service request when the Operation_Complete bit is set. (*ESE should be sent with a value of 1 and *SRE should be sent with a value of 32.)
4. Enable the controller to respond to the service request.
5. Send the command that requires control of the bus followed by the *OPC command.
6. Pass control to the analyzer and wait for the service request. The service request indicates that the command has been completed and control has been passed back to the controller.

NOTE For this procedure to work properly, no overlapped commands should be pending except the command that requires control of the bus. For more information on overlapped commands, see "Synchronization" in this chapter.

Chapter 6, "Programming Examples," contains an example program that passes control to the analyzer. In the example, control is passed so the analyzer can print the contents of its screen.

Chapter 3
Programming with Hierarchical Commands

Introduction

The HP 35660A's device-specific HP-IB commands are derived from elements of a command hierarchy (or tree). This chapter describes the command tree and discusses special characteristics of the analyzer's hierarchical commands. The chapter also describes the general syntax for program messages, which are used to send these commands to the analyzer. Finally, the chapter describes the general syntax for response messages, which the analyzer uses to return data to other devices on the HP-IB.

The Command Tree

The HP 35660A's command tree organizes related analyzer functions by grouping them together on a common branch. Each branch is assigned a mnemonic to indicate the nature of the related functions. For example, the analyzer's marker functions are grouped together on the MARKER branch of the command tree. The MARKER branch is only one of 25 major branches on the tree. The other 24 branches organize the remaining device-specific functions. The branches are also referred to as subsystems.

When many device functions are grouped together on a particular branch, additional branching is used to organize device functions into groups that are even more closely related. The MARKER branch serves as a good example because the analyzer provides many marker functions. (See Appendix C for a complete list of commands that access marker functions.) Band marker functions are grouped together on the BAND branch of the MARKER branch, harmonic marker functions are grouped together on the HARMONIC branch, and so on.

The branching process continues until each analyzer function is assigned to its own branch. For example, the function that turns the analyzer's harmonic markers on and off is assigned to the STATE branch of the HARMONIC branch of the MARKER branch.

The command that accesses a particular function is created by:

1. Concatenating the mnemonics on a direct path from the base of the tree to the function's branch.
   For example, MARKERHARMONICSTATE
   

2. Separating the mnemonics with colons to indicate branching points on the tree.
   For example, MARKER:HARMONIC:STATE

3. Appending the value you want assigned to the function.
   For example, MARKER:HARMONIC:STATE ON

In Appendix C, steps 1 and 2 have already been completed for each command. Also, a particular command's position in the command tree is indicated both by colons and by levels of indentation. After completing step 3 for one of these commands, you can send it to the analyzer via your controller's language system. If you are using HP BASIC, for example, you could send:

```
OUTPUT 711;"MARKER:HARMONIC:STATE ON"
```

Sending Multiple Commands

You can send multiple commands within a single program message by separating the commands with semicolons (;). For example, the following program message — sent within an HP BASIC statement — would turn the harmonic markers on and set the number of harmonic markers to 3:

```
OUTPUT 711;"MARKER:HARMONIC:STATE ON;:MARKER:HARMONIC:COUNT 3"
```

The analyzer's command parser allows you to simplify the previous program message. This is because one of the parser's main functions is to keep track of a program message's position in the command tree. If you take advantage of this parser function, you can create the equivalent, but simpler, program message:

```
OUTPUT 711;"MARKER:HARMONIC:STATE ON;COUNT 3"
```

In the first version of the program message, the semicolon that separates the two commands is followed by a colon. Whenever this occurs, the command parser is reset to the base of the command tree. As a result, the next command is only valid if it includes the entire mnemonic path from the base of the tree.

In the second version of the program message, the semicolon that separates the two commands is not followed by a colon. Whenever this occurs, the command parser assumes that the mnemonics of the second command arise from the same branch of the tree as the final mnemonic of the preceding command. STATE, the final mnemonic of the preceding command, arises from the MARKER:HARMONIC branch. So COUNT, the first mnemonic of the second command, is also assumed to arise from the MARKER:HARMONIC branch.

Here is a longer series of commands — again, sent within HP BASIC statements — that can be combined into a single program message:

```
OUTPUT 711;"AVERAGE:STATE ON"
OUTPUT 711;"AVERAGE:TYPE PEAK"
OUTPUT 711;"AVERAGE:COUNT 100"
OUTPUT 711;"AVERAGE:DISPLAY:RATE 20"
OUTPUT 711;"AVERAGE:DISPLAY:RATE:STATE ON"
```

The single program message would be:

```
OUTPUT 711;"AVERAGE:STATE ON;TYPE PEAK;COUNT 100;DISPLAY:RATE 20;RATE:STATE ON"
```

Command Abbreviation

Each command mnemonic has a long form and a short form. The short forms of the mnemonics allow you to send abbreviated commands. The mnemonics' short forms are created according to the following rules:

1. If the long form of the mnemonic has less than four characters, the short form is the same as the long form. For example, ARM remains ARM.
2. If the long form of the mnemonic has exactly four characters, the short form is the same as the long form. For example, USER remains USER.
3. If the long form of mnemonic has more than four characters and the fourth character is a consonant, the short form consists of the first four characters of the long form. For example, AVERAGE becomes AVER.
4. If the long form of mnemonic has more than four characters and the fourth character is a vowel, the short form consists of the first three characters of the long form. For example, LIMIT becomes LIM.

NOTE The syntax descriptions in Chapter 7 use upper-case characters to identify the short form of a particular mnemonic.

If the rules listed in this section are applied to the last program message in the preceding section, the statement:

```
OUTPUT 711;"AVERAGE:STATE ON;TYPE PEAK;COUNT 100;DISPLAY:RATE 20;RATE:STATE ON"
```

becomes:

```
OUTPUT 711;"AVER:STAT ON;TYPE PEAK;COUN 100;DISP:RATE 20;RATE:STAT ON"
```

Message Syntax

As mentioned in Chapter 2, the analyzer uses program messages and response messages to communicate with other devices on the HP-IB. This section uses syntax diagrams to describe the general syntax rules for both kinds of messages.

Conventions

The flow of syntax diagrams is generally from left to right. However, elements that repeat require a return path that goes from right to left. Any message that can be generated by following a diagram from its entry point to its exit point, in the direction indicated by the arrows, is valid.

Angle brackets < > enclose the names of syntactic items that need further definition. The definition is included either in the text accompanying the diagram, in a subsequent diagram, or in the next section, "Common Definitions."

The symbol ::= means "is defined as." When two items are separated by this symbol, the second item can replace the first in any statement that contains the first item.

Common Definitions

The syntax diagrams have the following definitions in common:

- <LF> is the line feed character (ASCII decimal 10).
- <^END> is assertion of the HP-IB END message while the last byte of data is on the bus.
- <SP> is the space character (ASCII decimal 32).
- <WSP> is one or more white space characters (ASCII decimal 0-9 and 11-32).
- <digit> is one character in the range 0-9 (ASCII decimal 48-57).
- <alpha> is one character of the alphabet. The character can be either upper-case (ASCII decimal 65-90) or lower-case (ASCII decimal 97-122) unless otherwise noted.

Special Syntactic Elements

Several syntactic elements have special meanings. They are:

- (colon) : — When a command or query contains a series of mnemonics, the mnemonics are separated by colons. A colon immediately following a mnemonic tells the command parser that the program message is proceeding to the next level of the command tree. A colon immediately following a semicolon tells the command parser that the program message is returning to the base of the command tree. For more information, see "The Command Tree" and "Sending Multiple Commands" at the beginning of this chapter.

- (semicolon) ; — When a program message contains more than one command or query, a semicolon is used to separate them from each other. For example, if you want to autorange the analyzer's inputs and then start a measurement using one program message, the message would be:

```
INPUT:RANGE:AUTO ON;:INITIALIZE:STATE START
```

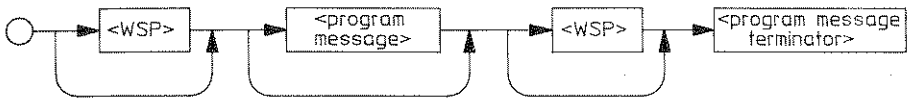
- (comma) , — A comma separates the data sent with a command or returned with a response. For example, the SYSTEM:TIME command requires three values to set the analyzer's clock: one for hours, one for minutes, and one for seconds. A message to set the clock to 8:45 AM would be:

```
SYSTEM:TIME 8,45,0
```

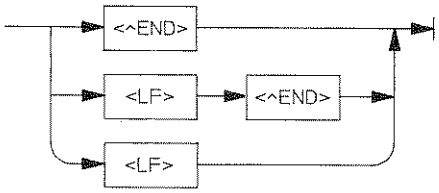
- <WSP> — One or more white space characters are optional in many parts of a program message. However, at least one is required to separate a command or query from the data sent with that command or query. The previous example contains a space between the command (SYSTEM:TIME) and the data sent with the command (8,45,0).
- <message terminator> — A message terminator is required at the end of a program message or a response message. Program message terminators are described in "Program Message Syntax." Response terminators are described in "Response Message Syntax."

Program Message Syntax

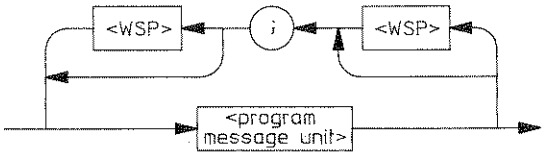
The syntax for a terminated program message is:



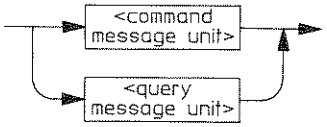
<program message terminator>::=



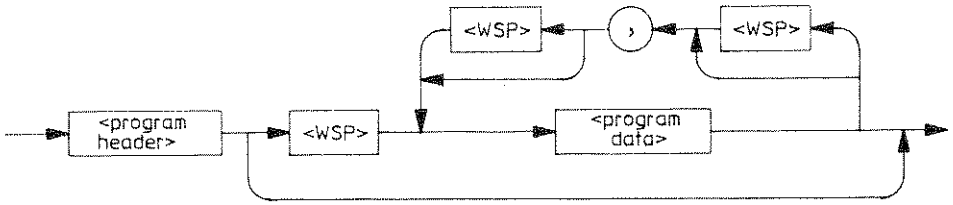
<program message>::=



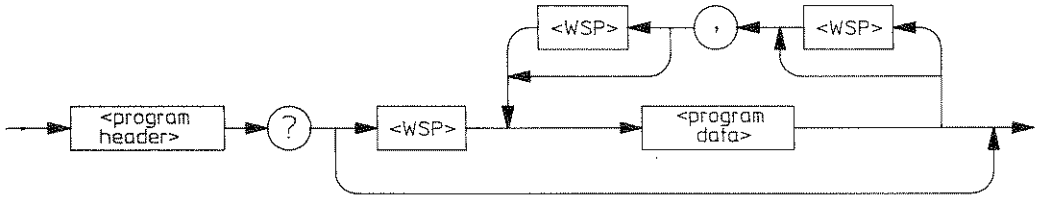
<program message unit>::=



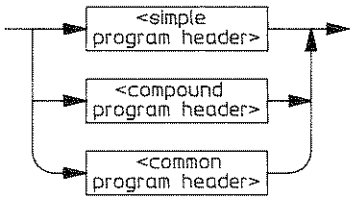
<command message unit>::=



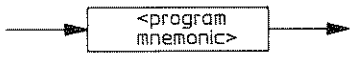
<query message unit>::=



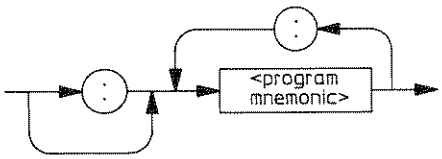
<program header>::=



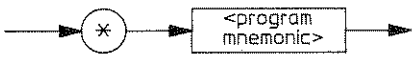
<simple program header>::=



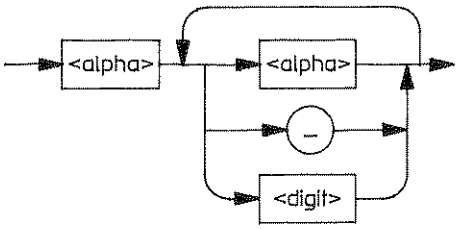
<compound program header>::=



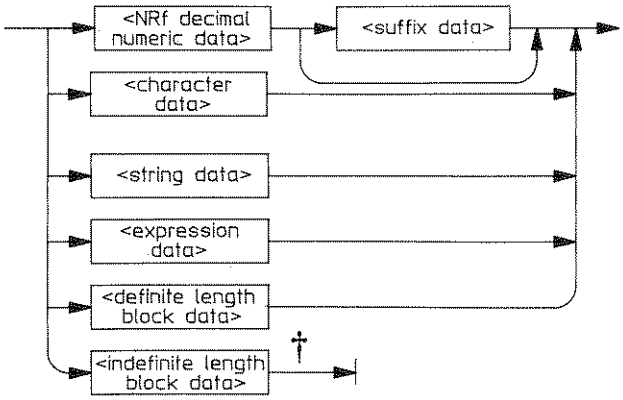
<common program header>::=



<program mnemonic>::=



<program data>::=



† The definition of indefinite length block data includes termination with <LF><^END>. This serves the dual function of terminating the data and terminating the program message.

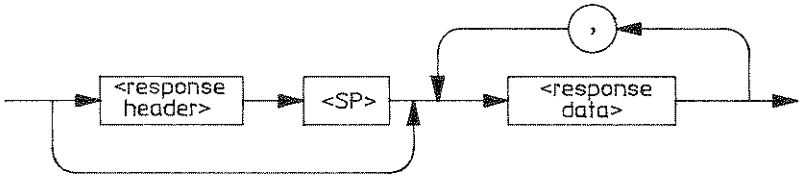
Program data and response data are described in Chapter 4, "Transferring Data."
<suffix data> is dependent on the command sent.

Response Message Syntax

The syntax for a terminated response message is:

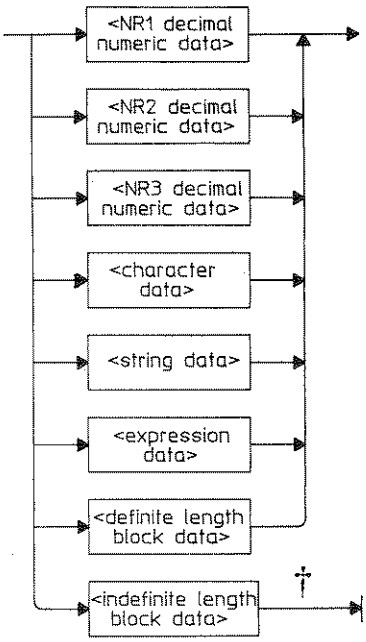
<response message terminator>::=

<response message>::=



<response header> is the <program header> sent with the query that generated the response. It is only returned as part of the <response message> if SYSTEM:HEADER is ON. Also, all <alpha> characters in the <response header> will be upper-case, even if they were lower-case in the <program header>.

<response data>::=



† The definition of indefinite length block data includes termination with <LF><^END>. This serves the dual function of terminating the data and terminating the response message.

Response data and program data are described in Chapter 4, "Transferring Data."

Chapter 4
Transferring Data

Data can be transferred between the analyzer and a controller via the HP-IB data lines, DIO1 through DIO8. Such transfers occur in a byte-serial (one byte at a time), bit-parallel (8 bits at a time) fashion. This chapter discusses:

- The different ways data bytes can be encoded
- The formats used to transfer different types of data
- The structure of HP 35660A files

Data Encoding

Two kinds of data encoding are used when data is transferred between the HP 35660A and an HP-IB controller: ASCII encoding, and binary encoding. All device commands and queries are sent over the HP-IB as a series of ASCII-encoded bytes. In most cases, the data sent with a command or returned in response to a query is also ASCII-encoded. However, when a large block of data is transferred, it can be either ASCII-encoded or binary-encoded.

Each command that is used to transfer block data has an associated command that allows you to select data encoding. The commands for selecting data encoding use the parameter ASC to specify ASCII encoding. They use either BIN, FP32, or FP64 to specify binary encoding. For example, SYST:SET transfers a block of data that defines the instrument state. The command SYST:SET:FORM allows you to specify encoding for the data in the block. Another command, TRAC:DATA, transfers a block of trace data. In this case, the command TRAC:HEAD:AFOR allows you to specify encoding for the data in the block.

ASCII Encoding

Most data that is transferred between the analyzer and an HP-IB controller is encoded using the ASCII 7-bit code (defined by the ANSI X3.4-1977 standard). When an ASCII-encoded byte is sent over the bus, bits 1 through 7 of the byte (bit 1 being the least significant bit) correspond to the HP-IB data lines DIO1 through DIO7. DIO8 is ignored. The formats used for ASCII-encoded data are discussed in "Data Formats" later in this chapter.

Binary Encoding

Binary encoding can only be used for block data. In addition, binary encoding can only be used for the numeric fields in block data. For example, SYST:SET transfers a block of data that defines the instrument state. Many of the fields in the block contain character data. The character data is always ASCII-encoded, even when you have specified binary encoding.

Numeric fields in block data can contain either integers, fixed point numbers, or floating point numbers. A binary-encoded integer format is used to represent integers when binary encoding is specified. A binary floating point format is used to represent fixed and floating point numbers when binary encoding is specified.

Binary-Encoded Integers

Binary-encoded integers can be one, two, or four bytes long. The most significant byte of two and four byte integers is always sent over the HP-IB first. The order of the bits corresponds to the order of the HP-IB data lines. The most significant bit corresponds to DIO8 and the least significant bit corresponds to DIO1. Data is right justified and in two's complement notation. The most significant bit is used as the sign bit.

For example, in a two-byte integer, 7 (decimal) would be encoded as:

Binary Floating Point Numbers

When binary encoding is specified, the 32-bit and 64-bit binary floating point formats defined in the IEEE 754-1985 standard are used to represent both fixed and floating point decimal numbers. Many controllers, and the languages that run on them, use these formats. Both formats have three fields in common, but the length of the fields are different for each. The fields and their bit lengths appear in the following table:

Table 4-1. Fields in Binary Floating Point Numbers

Field	Width of Field	
	32-bit format	64-bit format
sign (s)	1 bit	1 bit
exponent (e)	8	11
fraction (f)	23	52

When the 32-bit format is used, the decimal value of the exponent field ranges from −126 to +127, with a bias of +127. When the 64-bit format is used, the decimal value of the exponent field ranges from −1022 to +1023, with a bias of +1023.

You can use the following formulas to determine the value (x) of a 32-bit binary floating point number. (s, e, and f must be converted from binary to decimal before using the formulas.)

If e = 255 and f ≠ 0	then x is not a number
If e = 255 and f = 0	then $x = -1^{s}(\infty)$
If 0 < e < 255	then $x = -1^{s}(2^{e-127})(1 + f)$
If e = 0 and f ≠ 0	then $x = -1^{s}(2^{e-126})(0 + f)$
If e = 0 and f = 0	then $x = -1^{s}(0)$

32-bit binary floating point numbers are sent over the bus as follows:

	DIO 8 7 6 5 4 3 2 1
byte 1	s e e e e e e e
byte 2	e f f f f f f f
bytes 3 and 4	f f f f f f f f

You can use the following formulas to determine the value (x) of a 64-bit binary floating point number. (Again, s, e, and f must be converted from binary to decimal before using the formulas.)

If e = 2047 and f ≠ 0	then x is not a number
If e = 2047 and f = 0	then $x = -1^{s}(\infty)$
If 0 < e < 2047	then $x = -1^{s}(2^{e-1023})(1 + f)$
If e = 0 and f ≠ 0	then $x = -1^{s}(2^{e-1022})(0 + f)$
If e = 0 and f = 0	then $x = -1^{s}(0)$

64-bit binary floating point numbers are sent over the bus as follows:

	DIO 8 7 6 5 4 3 2 1
byte 1	s e e e e e e e
byte 2	e e e e f f f f
bytes 3 through 8	f f f f f f f f

Here is an example of a number encoded in the 32-bit binary floating point format:

byte 1	byte 2	byte 3	byte 4
01000001	10010000	00000000	00000000
seeeeeee	efffffff	ffffffff	ffffffff

Where:

	binary	decimal
s =	0	= 0
e =	10000011	= 131
f =	.001	= .125

Therefore:

$$x = (-1)^0(2^{(131-127)})(1.125)$$
$$= (2^4)(1.125)$$
$$= 18$$

Data Formats

The HP 35660A uses a number of different data formats to represent the different types of data it uses. The formats are described in this section using syntax diagrams.

Conventions

The flow of syntax diagrams is generally from left to right. However, elements that repeat require a return path that goes from right to left. Any data you can generate by following a diagram from its entry point to its exit point, in the direction indicated by the arrows, is valid.

Angle brackets < > enclose the names of syntactic items that need further definition. The definition is included either in the text accompanying the diagram, or in the next section, "Common Definitions."

Common Definitions

The syntax diagrams have the following definitions in common:

- < LF > is the line feed character (ASCII decimal 10).
- < ^END > is assertion of the HP-IB END message while the last byte of data is on the bus.
- < SP > is the space character (ASCII decimal 32).
- < WSP > is one or more white space characters (ASCII decimal 0-9 and 11-32).
- < digit > is one character in the range 0-9 (ASCII decimal 48-57).
- < non-zero digit > is one character in the range 1-9 (ASCII decimal 49-57).
- < alpha > is one character of the alphabet. The character can be either upper-case (ASCII decimal 65-90) or lower-case (ASCII decimal 97-122) unless otherwise noted.

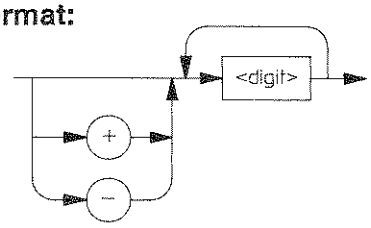
Decimal Numeric Data

The analyzer returns three types of decimal numeric data in response to queries:

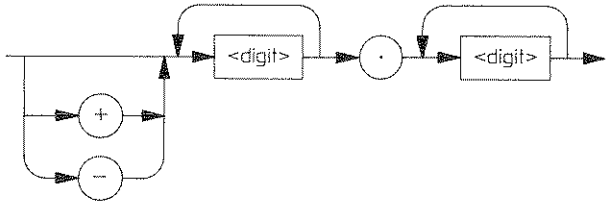
- Integers — returned using the NR1 format
- Fixed point numbers— returned using the NR2 format
- Floating point numbers — returned using the NR3 format

You can use a more flexible format, the NRf format, when sending any of the three decimal numeric data types to the analyzer. All four formats are described in the following syntax diagrams.

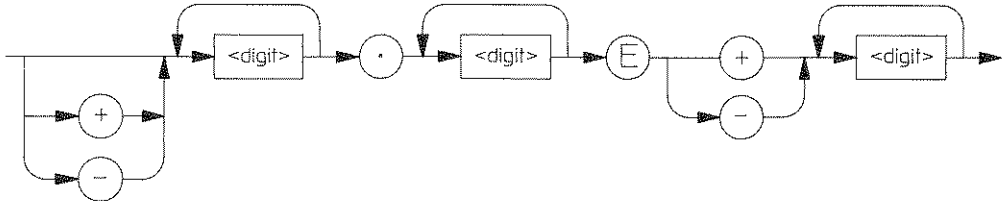
NR1 format:



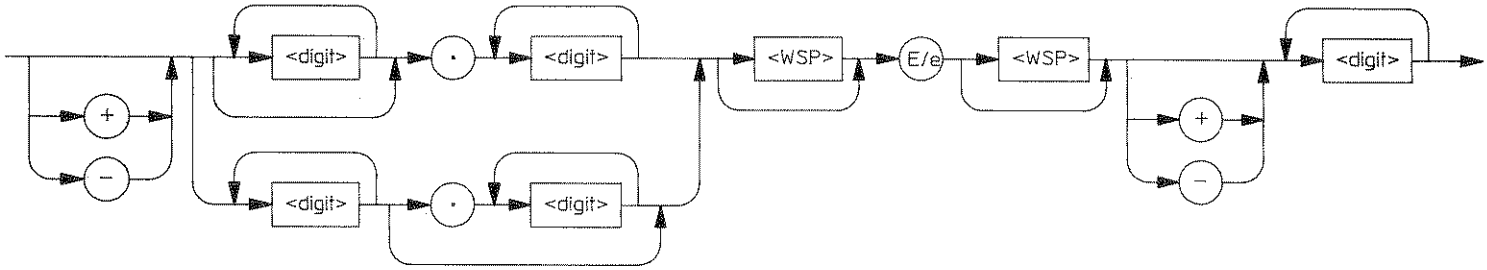
NR2 format:



NR3 format:

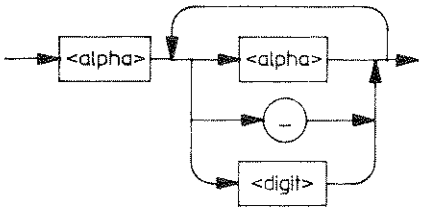


NRf format:



Character Data

The format you use to send character data is:

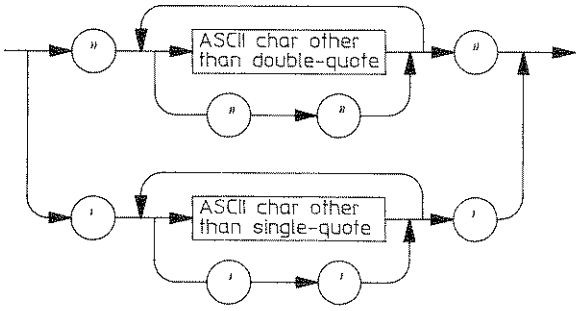


The "_" in the circle is the underscore character (ASCII decimal 95).

The format used when the analyzer returns character data is the same as the format used to send character data, with one exception — the analyzer never returns lower-case alpha characters.

String Data

The format you use to send string data is:

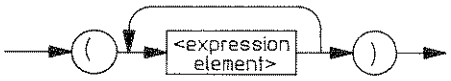


Note that you must use two double-quote characters ("") to represent one (") in a string that is delimited by double-quote characters. You must use two single-quote characters ('') to represent one (') in a string that is delimited by single-quote characters.

The format used when the analyzer returns string data is the same as the format used to send string data, with one exception — the analyzer never returns string data using the single-quote path.

Expression Data

The format you use to send expression data is:



The the only command that uses expression data is USER:EXPR. The syntax description for that command contains a list of acceptable < expression elements >.

The format used when the analyzer returns expression data is the same as the format used to send expression data.

Block Data

The analyzer typically uses one of two block data formats to send or receive large amounts of data:

- The definite length block format
- The indefinite length block format

The definite length format is used when binary encoding has been specified for the block. The indefinite length format is usually used when ASCII encoding has been specified. However, some commands that send block data simply use one of the decimal numeric data formats (NR1-3 or NRf) when ASCII encoding has been specified. In these cases, the block is sent as a series of NRx numbers separated by commas.

Definite Length Block Data

The format you use to send definite length block data is:

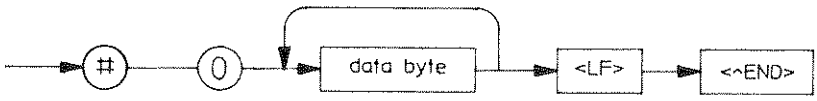
The elements #, < non-zero digit >, and < digit > make up a header for the block data. < non-zero digit > indicates how many times < digit > is repeated. The < digits > are interpreted as a single decimal number, which indicates how many bytes of data follow in the block. Here is an example:

Block Header				Block Data			
byte 1	byte 2	byte 3	byte 4	byte 5	byte 6	...	byte 19
#	2	1	5	< data_byte_1 >	< data_byte_2 >	...	< data_byte_15 >

< non-zero digit > is 2, which means that the following two bytes should be taken together as a single decimal number. In this case, the number is 15. The following 15 bytes are the 5^{th} through 19^{th} bytes of the data transfer, but they are the 1^{st} through 15^{th} bytes of the data block.

Indefinite Length Block Data

The format you use to send indefinite length block data is:



The first two bytes of the data transfer, # and 0, make up a header for the block data. The data itself does not begin until the third byte of the data transfer.

File Formats

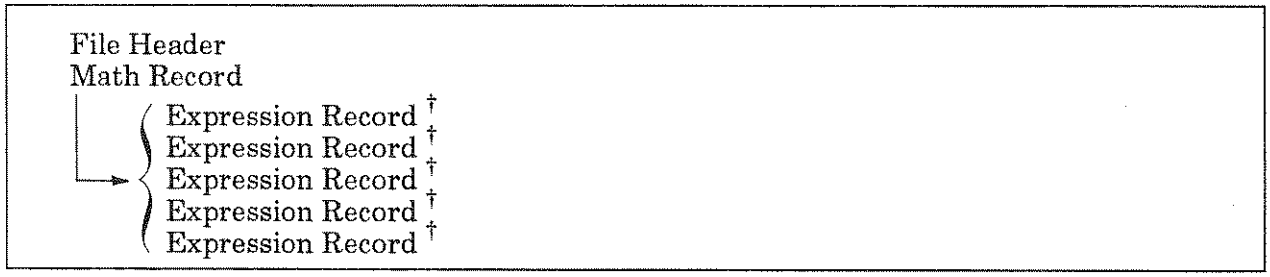
The HP 35660A can save and recall five different file types. They are:

- The math file
- The limit table file
- The data table file
- The instrument state file
- The trace file

This section describes each of the file types.

Basic File Structures

The following five figures show you the basic structure of the different file types. Each file is made up of a file header followed by one or more records. The records are arranged in a hierarchical fashion. The hierarchy can be thought of as a series of parent/child relationships. Arrows in the figures point from a parent record to each of its child records. The file header and all of the records are described in tables at the end of this chapter.

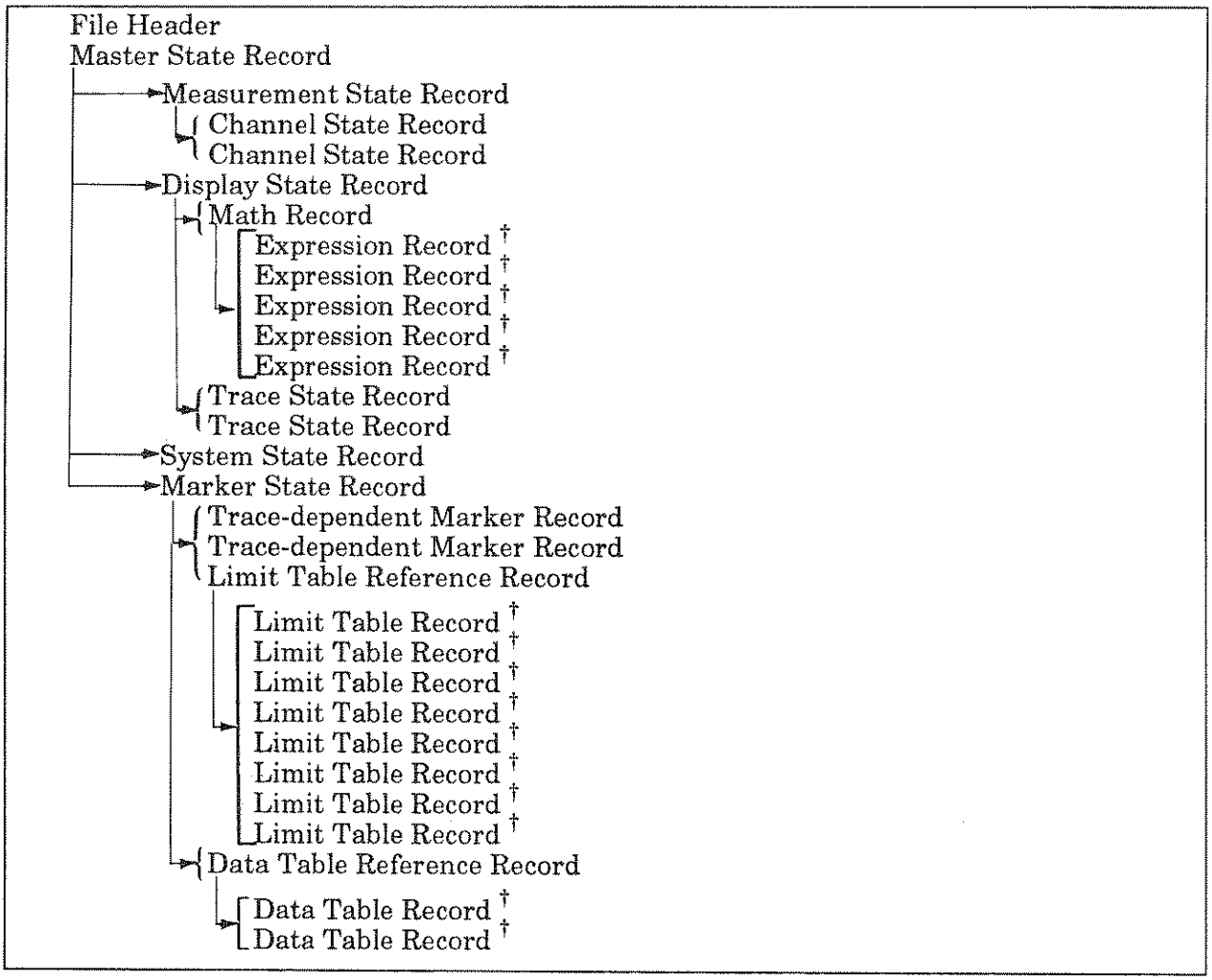


† This record is optional.

Figure 4-1. Math File Structure

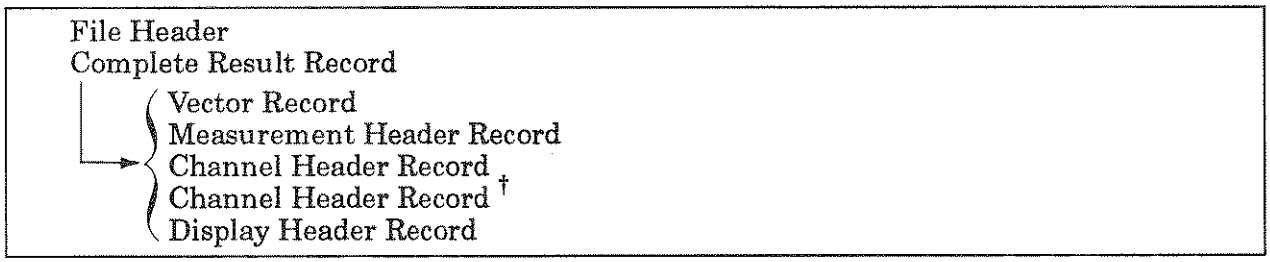
Figure 4-2. Limit Table File Structure

Figure 4-3. Data Table File Structure



† This record is optional.

Figure 4-4. Instrument State File Structure



† This record is optional.

Figure 4-5. Trace File Structure

Special Fields in a Record

All records contain the following three special fields:

- Record Type
- Record Length
- Total Record Reference Count

If a record is a parent record, it contains at least one additional special field called a Record Reference.

Record Type

The Record Type field contains a number that uniquely identifies the type of data contained in the record. For example, all records with a Record Type number of 100,794,368 contain HP 35660A channel header data. The fields in any two such records are the same, but the values in each record's fields may be different. For example, one Channel Header record may contain channel 1 header data, while another Channel Header record contains channel 2 header data.

Record Length

The Record Length field contains a number that specifies the length of the record. If the record is ASCII-encoded, the length is specified as a number of lines. If the record is binary-encoded, the length is specified as a number of bytes.

NOTE The length of a binary-encoded record is always a multiple of 128. If the number of bytes required by a record's fields is not divisible by 128, then the record is padded with zeros to the nearest multiple of 128.

Total Record Reference Count

The Total Record Reference Count field contains a number that specifies how many child records are referenced from the record. If the number is 0, then the record is not a parent record. If the number is greater than 0, then the record is a parent record and will contain at least one Record Reference field.

Record Reference

All parent records contain Record Reference fields. It is these fields that link individual records together to form a particular file type. A Record Reference forms the links by:

- Identifying the type of child record that follows the parent record
- Indicating how many times that child record is repeated in the file

The position of a Record Reference in a parent record determines the identity of the child record. For example, line 4 of the Math Record (see Table 4-2) is used as a Record Reference for Expression Records. The value assigned to a Record Reference determines how many times the record is repeated in the file. Using the Math Record again, if the value of line 4 is 2, then two Expression Records will follow the Math Record.

ASCII Index	Binary Index	Meaning of Data/ Data	Data Type	Range/ Units
line 1	byte 1:4	RECORD TYPE	long	393216 (00060000 when converted to hexadecimal)
2	5:8	RECORD LENGTH: ASCII or Binary	long	14 lines or128 bytes
3	9:12	TOTAL RECORD REFERENCE COUNT	long	0:5
4	13:16	RECORD REFERENCE: # of expression records to follow	long	0:5
5	17:24	real part of constant K1	double	−340.28E+36:340.28E+36
6	25:32	Imaginary part of constant K1	double	−340.28E+36:340.28E+36

Table 4-2. Partial Math Record Description

The Order of Records in a File

Two rules determine the order of records in a file.

1. Child records always follow immediately after their parent records.
2. If there are two or more Record References in a parent record, their order determines the order of the child records.

An example will help to illustrate these rules. Suppose that the master record (Record type M) of a file contains the following lines:

line	description	value
1	RECORD TYPE	0
2	RECORD LENGTH	6
3	TOTAL RECORD REFERENCE COUNT	4
4	RECORD REFERENCE: Record type X	1
5	RECORD REFERENCE: Record type Y	2
6	RECORD REFERENCE: Record type Z	1

The order of records in the file is:

Record 1 (type M)
- Record 2 (type X)
- Record 3 (type Y)
- Record 4 (type Y)
- Record 5 (type Z)

Example File

The following illustration (Figure 4-6) shows the contents of an example Math file. The file contains a File Header, a Math record, and two Expression records.

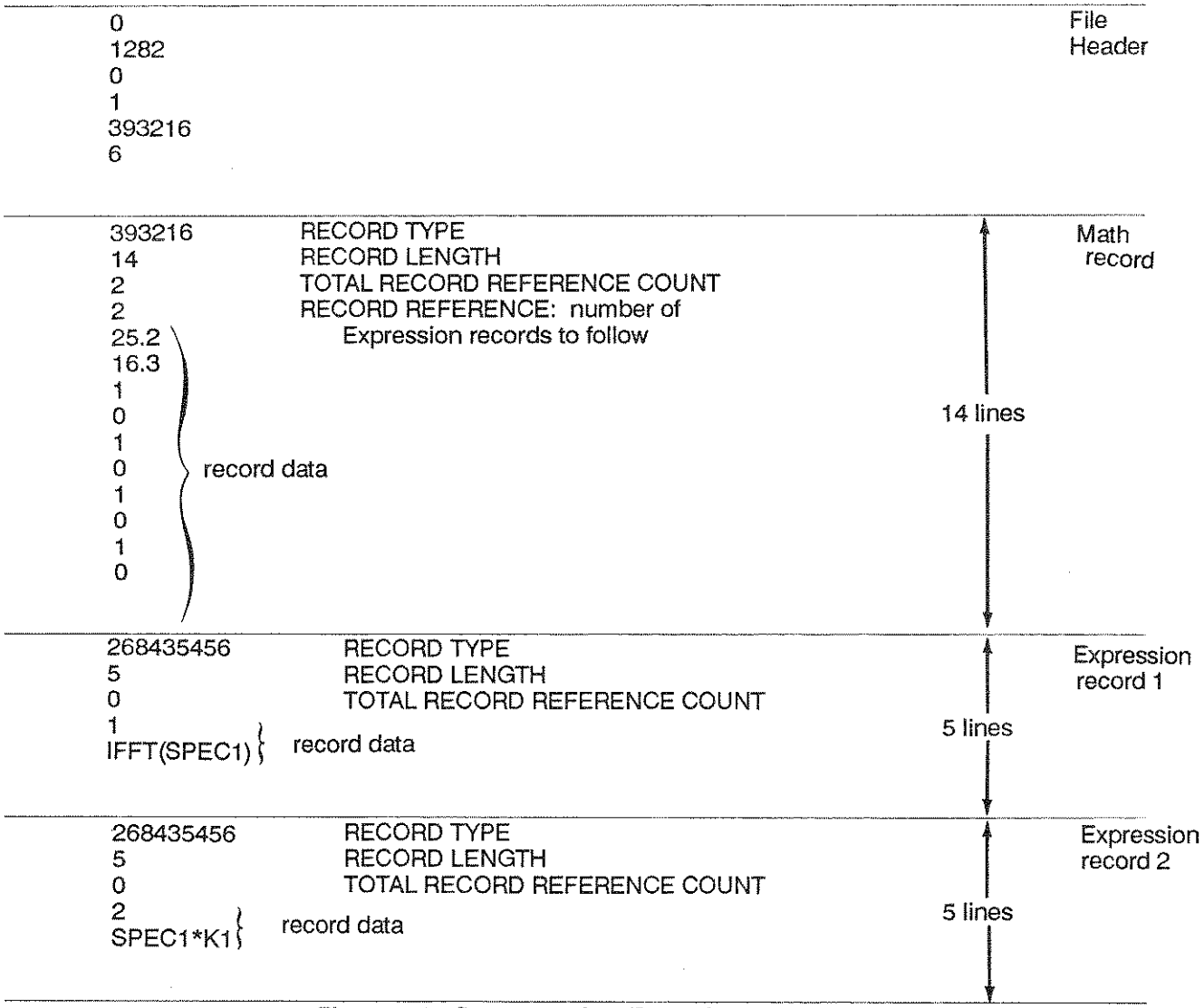


Figure 4-6. Contents of an Example Math File

Controller Access to Files

Two HP-IB commands give a controller direct access to the information available in HP 35660A files. The commands are:

- SYST:SET
- TRAC:DATA:SET

SYST:SET provides direct access to the instrument state file structure. Figure 4-4 shows the structure of transferred data with one exception — the file header is not used for direct transfers between the analyzer and a controller. TRAC:DATA:SET provides direct access to the trace file structure. Figure 4-5 shows the structure of transferred data — again, with the exception that the file header is not used for direct transfers between the analyzer and a controller.

Record Descriptions

The following tables describe the individual records from which the analyzer's five file types are built. Each table includes:

- A description of the individual fields in the record
- The acceptable range of values for data in the field
- A binary and an ASCII index for each field
- An indication of the type of data used in each field

Fields require both binary and ASCII indexes because of differences related to data encoding. When a file is binary-encoded, each field is assigned a fixed number of bytes. So the index into a particular field of a binary-encoded file is a range of byte numbers. When a file is ASCII-encoded, the number of bytes in a field is variable. However, all fields are separated by line feed characters (ASCII decimal 10). So the index into a particular field of an ASCII-encoded file is a line number.

The file data types are as follows:

- Char[n] – This data type consists of a series of ASCII-encoded bytes. When the whole file is ASCII-encoded, [n] specifies the maximum number of bytes in the field. When the whole file is binary-encoded, [n] specifies the actual number of bytes in the field.
- Bool – This data type has acceptable values of 0 and 1. When the file is ASCII-encoded, the value is simply transferred as an ASCII-encoded 0 or 1. When the file is binary-encoded, the 0 or 1 is transferred as a one-byte, binary-encoded integer.
- Short – This data type is used for integers (maximum range of −32768 to +32767). When the file is ASCII-encoded, values are transferred using the NR1 format. When the file is binary-encoded, values are transferred as two-byte, binary-encoded integers.
- Long – This data type is used for integers (maximum range of −2,147,483,648 to +2,147,483,647). When the file is ASCII-encoded, values are transferred using the NR1 format. When the file is binary-encoded, values are transferred as four-byte, binary-encoded integers.
- Float – This data type is used for single-precision fixed point and floating point numbers. When the file is ASCII-encoded, values are transferred using either the NR1, NR2, or NR3 format. When the file is binary-encoded, values are transferred using the 32-bit binary floating point format.
- Double – This data type is used for double-precision fixed point and floating point numbers. When the file is ASCII-encoded, values are transferred using either the NR1, NR2, or NR3 format. When the file is binary-encoded, values are transferred using the 64-bit binary floating point format.
- E-short – This data type is simply a short whose value is encoded. The meaning assigned to each value is included as part of the field's description.

Table 4-3 . Channel Header Record

ASCII Index	Binary Index	Meaning of Data/ Data	Data Type	Range/ Units
line 1	byte 1:4	RECORD TYPE	long	100794368 (06020000 when converted to hexadecimal)
2	5:8	RECORD LENGTH: ASCII or Binary	long	12 lines or 128 bytes
3	9:12	TOTAL RECORDS REFERENCED	long	0
4	13:20	time-record overlap per average (requested)	double	0:0.99
5	21:22	overloads occurred during measurement (0=no, 1=yes)	short	0:1
6	23:24	input coupling (1=DC, 2=AC)	e-short	1:2
7	25:32	input range	double	−51:27 dBVrms
8	33:34	windowing function 1 = Uniform 2 = Hanning 3 = Flat Top 4 (not supported) 5 = force 6 = exponential	e-short	1:6
† 9	35:50	name of channel providing this vector's data	char[16]	
10	51:58	force/exponential window decay	double	0:1E+100 s
11	59:66	force window width	double	0:1E+100 s
12	67:68	input grounding (0=grounded, 1=floating)	e-short	0:1
—	69:128	padding (binary file only)		

†This field is ignored on recall.

Table 4-4. Channel State Record

ASCII Index	Binary Index	Meaning of Data/ Data	Data Type	Range/ Units
line 1	byte 1:4	RECORD TYPE	long	218103808 (0d000000 when converted to hexadecimal)
2	5:8	RECORD LENGTH: ASCII or Binary	long	18 lines or 128 bytes
3	9:12	TOTAL RECORD REFERENCE COUNT	long	0
4	13:14	# of the channel this record describes	short	1:2
5	15:16	windowing function 0 = Uniform 1 = Hanning 2 = Flat Top 3 (not supported) 4 = force/exponential	e-short	0:4
6	17	force/exponential type (0=force, 1=exponential)	bool	0:1
7	18:25	force/exponential window decay	double	0:1E+100 s
8	26:33	force window width	double	0:1E+100 s
9	34:41	input range	double	−51:27 dBVrms
10	42	auto-range state (0=off, 1=on)	bool	0:1
11	43	input grounding (0=grounded, 1=floating)	bool	0:1
12	44	input coupling (0=DC, 1=AC)	bool	0:1
13	45	engineering units state (0=volts active, 1=EUs active)	bool	0:1
14	46:53	volts to eng. units conversion factor	double	−1E+100:1E+100, except 0
15	54:69	eng. units label for input range	char[16]	upper- and lower-case alpha, numbers, spaces, and ;:,.+−*^/_\()[]{}
16	70:77	dBVrms to normal units conversion factor	double	(dependent on normal units)
17	78:93	active normal units label for input range	char[16]	dBVrms, dBVpk, Vrms, V, dBm
18	94:101	trigger delay	double	(range dependent on span) s
—	102:128	padding (binary file only)		

Table 4-5 . Complete Result Record

ASCII Index	Binary Index	Meaning of Data/ Data	Data Type	Range/ Units
line 1	byte 1:4	RECORD TYPE	long	262144 (00040000 when converted to hexadecimal)
2	5:8	RECORD LENGTH: ASCII or Binary	long	7 lines or 128 bytes
3	9:12	TOTAL RECORDS REFERENCED	long	4:5
4	13:16	RECORD REFERENCE: # of vector records to follow	long	1
5	17:20	RECORD REFERENCE: # of measurement header records to follow	long	1
6	21:24	RECORD REFERENCE: # of channel header records to follow	long	1:2
7	25:28	RECORD REFERENCE: # of display header records to follow	long	1
—	29:128	padding (binary file only)		

Table 4-6. Data Table Record

ASCII Index	Binary Index	Meaning of Data/ Data	Data Type	Range/ Units
line 1	byte 1:4	RECORD TYPE	long	369098752 (16000000 when converted to hexadecimal)
2	5:8	RECORD LENGTH: ASCII or Binary	long	9:2056 lines or 128:16512 bytes
3	9:12	TOTAL RECORD REFERENCE COUNT	long	0
4	13:14	# of the data table this record describes	short	0:1
5	15	data table calculation (0=off, 1=on)	bool	0:1
6	16:17	# of points in the table	short	1:401
7	18:21	skip from record start to data start	long	7 lines or 21 bytes
Data table data starts here. It consists of a number of points that are directly adjacent to each other in the record. Each point is defined by 2 values. Points are repeated n times to a maximum of 401 points. A point takes this form:				
8	22:25	x-axis value	float	−120E+3:120E+3 Hz or s
9	26:29	y-axis value	float	(dependent on vertical units)
—	—	points 2 through n		
—	—	padding to multiple of 128 (binary file only)		

Table 4-7. Data Table Reference Record

ASCII Index	Binary Index	Meaning of Data/ Data	Data Type	Range/ Units
line 1	byte 1:4	RECORD TYPE	long	524288 (00080000 when converted to hexadecimal)
2	5:8	RECORD LENGTH: ASCII or Binary	long	4 lines or 128 bytes
3	9:12	TOTAL RECORD REFERENCE COUNT	long	0:2
4	13:16	RECORD REFERENCE: # of data table records to follow	long	0:2
—	17:128	padding (binary file only)		

Table 4-8 . Display Header Record

ASCII Index	Binary Index	Meaning of Data/ Data	Data Type	Range/ Units
line 1	byte 1:4	RECORD TYPE	long	167837696 (0A010000 when converted to hexadecimal)
2	5:8	RECORD LENGTH: ASCII or Binary	long	21 lines or256 bytes
3	9:12	TOTAL RECORDS REFERENCED	long	0
4	13:14	type of coordinates 0 = linear magnitude 1 = logarithmic magnitude 2 = magnitude 3 = phase 4 = real 5 = imaginary 6 = group delay 7 = user math 8 = null	e-short	0:8
5	15	data labelling (0=off, 1=on)	bool	0:1
6	16	x-axis scaling (0=linear, 1=logarithmic)	bool	0:1
7	17:32	y-axis engineering units label	char[16]	upper- and lower-case alpha, numbers, spaces, and ;:,.+−*^/_\()[]{}
8	33:40	internal to eng. units conversion factor (for y-axis label)	double	−1E100:1E+100, except 0
9	41	data valid (0=no, 1=yes)	bool	0:1
10	42:43	reserved	short	0
11	44	reference level tracking (0=off,1=on)	bool	0:1
12	45:60	internal units label (for y-axis per division)	char[16]	V, V^2, Vrms, Vrms^2, dB, deg, rad, V/rtHz, Vrms/rtHz, V^2/Hz, Vrms^2/Hz, s
13	61:76	engineering units label (for y-axis per div.)	char[16]	upper- and lower-case alpha, numbers, spaces, and ;:,.+−*^/_\()[]{}
14	77:84	internal to eng. units conversion factor (for y-axis per div.)	double	−1E+100:1E+100, except 0
15	85:92	per div. value for y-axis (in internal units)	double	(dependent on per div. units)
16	93:108	internal units label (for y-axis reference)	char[16]	V, V^2, Vrms, Vrms^2, dB, dBm, dBVpk, dBVrms, deg, rad, V/rtHz, Vrms/rtHz, V^2/Hz, Vrms^2/Hz, dBVpk/Hz, dBVrms/Hz, dBm/Hz, s
17	109:124	engineering units label (for y-axis ref.)	char[16]	upper- and lower-case alpha, numbers, spaces, and ;:,.+−*^/_\()[]{}
18	125:132	internal to eng. units conversion factor (for y-axis ref.)	double	−1E+100:1E+100, except 0
19	133:140	value for top of y-axis	double	(dependent on y-axis ref. units)
20	141:148	value for center of y-axis	double	(dependent on y-axis ref. units)
21	149:156	value for bottom of y-axis	double	(dependent on y-axis ref. units)
—	157:256	padding (binary file only)		

Table 4-9. Display State Record

ASCII Index	Binary Index	Meaning of Data/ Data	Data Type	Range/ Units
line 1	byte 1:4	RECORD TYPE	long	234881024 (0E000000 when converted to hexadecimal)
2	5:8	RECORD LENGTH: ASCII or Binary	long	9 lines or 128 bytes
3	9:12	TOTAL RECORD REFERENCE COUNT	long	3
4	13:14	display format 0 = upper/lower 1 = single 2 = front/back 3 = state display	e-short	0:3
5	15	frequency label blanking (0=off, 1=on)	bool	0:1
6	16	display blanking (0=off, 1=on)	bool	0:1
7	17:18	# of the active trace	short	1:2
8	19:22	RECORD REFERENCE: # of math records to follow	long	1
9	23:26	RECORD REFERENCE: # of trace state records to follow	long	2
—	27:128	padding (binary file only)		

Table 4-10. Expression Record

ASCII Index	Binary Index	Meaning of Data/ Data	Data Type	Range/ Units
line 1	byte 1:4	RECORD TYPE	long	268435456 (10000000 when converted to hexadecimal)
2	5:8	RECORD LENGTH: ASCII or Binary	long	5 lines or 384 bytes
3	9:12	TOTAL RECORD REFERENCE COUNT	long	0
4	13:14	# of the function (F1-F5) this record describes	short	1:5
5	15:284	expression that defines the function	char[270]	SPEC1, SPEC2, PSD1, PSD2, TIME1, TIME2, FRES, COH, CSP, F1:F5, K1:K5, '<filename>', JOM, CONJ, MAG, REAL, IMAG, SQRT, FFT, IFFT, and ()+−*/
—	285:384	padding (binary file only)		

Table 4-11. File Header

ASCII Index	Binary Index	Meaning of Data/ Data	Data Type	Range/ Units
line 1	byte 1:2	system ID	short	0
2	3:4	file encoding 1281=binary 1282=ASCII	e-short	1281, 1282
3	5:6	version number (major part)	short	0
4	7:8	version number (minor part)	short	1
5	9:12	file type 262144=trace file 327680=instrument state file 393216=math file 458752=limit table file 524288=data table file	long	262144, 327680, 393216, 458752, 524288
6	13:16	RECORD LENGTH: ASCII or binary	long	6 lines or 128 bytes
--	17:128	padding (binary file only)		

Table 4-12. Limit Table Record

ASCII Index	Binary Index	Meaning of Data/ Data	Data Type	Range/ Units
line 1	byte 1:4	RECORD TYPE	long	352321536 (15000000 when converted to hexadecimal)
2	5:8	RECORD LENGTH: ASCIIor Binary	long	13:10248 lines or 128:82048 bytes
3	9:12	TOTAL RECORD REFERENCE COUNT	long	0
4	13:14	# of the limit table this record describes	short	1:8
5	15	offset y-axis entry state (0=off, 1=on)	bool	0:1
6	16:19	offset y-axis entry value	float	(dependent on vertical units)
7	20:21	# of segments in the table	short	1:802
8	22:25	skip from record start to data start	long	8 lines or 25 bytes
Limit table data starts here. The data consists of a number of segments that are directly adjacent to each other in the record. Each segment is defined by five values. Segments are repeated n times to a maximum of 802 segments. A segment takes this form:				
9	26	upper or lower limit (0=lower, 1=upper)	bool	0:1
10	27:30	x-axis start value	float	−120E+3:120E+3 Hz or s
11	31:34	x-axis stop value	float	−120E+3:120E+3 Hz or s
12	35:38	y-axis start value	float	(dependent on vertical units)
13	39:42	y-axis stop value	float	(dependent on vertical units)
—	—	segments 2 through n		
—	—	padding to multiple of 128 (binary file only)		

Table 4-13. Limit Table Reference Record

ASCII Index	Binary Index	Meaning of Data/ Data	Data Type	Range/ Units
line 1	byte 1:4	RECORD TYPE	long	458752 (00070000 when converted to hexadecimal)
2	5:8	RECORD LENGTH: ASCII or Binary	long	4 lines or 128 bytes
3	9:12	TOTAL RECORD REFERENCE COUNT	long	0:8
4	13:16	RECORD REFERENCE: # of limit table records to follow	long	0:8
—	17:128	padding (binary file only)		

Table 4-14. Marker State Record

ASCII Index	Binary Index	Meaning of Data/ Data	Data Type	Range/ Units
line 1	byte 1:4	RECORD TYPE	long	318767104 (13000000 when converted to hexadecimal)
2	5:8	RECORD LENGTH: ASCII or Binary	long	6 lines or 128 bytes
3	9:12	TOTAL RECORD REFERENCE COUNT	long	4
4	13:16	RECORD REFERENCE: # of trace-dependent marker records to follow	long	2
5	17:20	RECORD REFERENCE: # of limit table reference records to follow	long	1
6	21:24	RECORD REFERENCE: # of data table reference records to follow	long	1
—	25:128	padding (binary file only)		

Table 4-15. Master State Record

ASCII Index	Binary Index	Meaning of Data/ Data	Data Type	Range/ Units
line 1	byte 1:4	RECORD TYPE	long	327680 (00050000 when converted to hexadecimal)
2	5:8	RECORD LENGTH: ASCII or Binary	long	9 lines or 128 bytes
3	9:12	TOTAL RECORD REFERENCE COUNT	long	4
4	13:16	time stamp (six-digit integer: hhmmss)	long	000000:235959
5	17:20	date stamp (six-digit integer: mmddyy)	long	010100:123199
6	21:24	RECORD REFERENCE: # of measurement state records to follow	long	1
7	25:28	RECORD REFERENCE: # of display state records to follow	long	1
8	29:32	RECORD REFERENCE: # of system state records to follow	long	1
9	33:36	RECORD REFERENCE: # of marker state records to follow	long	1
—	37:128	padding (binary file only)		

Table 4-16. Math Record

ASCII Index	Binary Index	Meaning of Data/ Data	Data Type	Range/ Units
line 1	byte 1:4	RECORD TYPE	long	393216 (00060000 when converted to hexadecimal)
2	5:8	RECORD LENGTH: ASCII or Binary	long	14 lines or128 bytes
3	9:12	TOTAL RECORD REFERENCE COUNT	long	0:5
4	13:16	RECORD REFERENCE: # of expression records to follow	long	0:5
5	17:24	real part of constant K1	double	−340.28E+36:340.28E+36
6	25:32	imaginary part of constant K1	double	−340.28E+36:340.28E+36
7	33:40	real part of constant K2	double	−340.28E+36:340.28E+36
8	41:48	imaginary part of constant K2	double	−340.28E+36:340.28E+36
9	49:56	real part of constant K3	double	−340.28E+36:340.28E+36
10	57:64	imaginary part of constant K3	double	−340.28E+36:340.28E+36
11	65:72	real part of constant K4	double	−340.28E+36:340.28E+36
12	73:80	imaginary part of constant K4	double	−340.28E+36:340.28E+36
13	81:88	real part of constant K5	double	−340.28E+36:340.28E+36
14	89:96	imaginary part of constant K5	double	−340.28E+36:340.28E+36
—	97:128	padding (binary file only)		

Table 4-17 . Measurement Header Record

ASCII Index	Binary Index	Meaning of Data/ Data	Data Type	Range/ Units
line 1	byte 1:4	RECORD TYPE	long	84017152 (05020000 when converted to hexadecimal)
2	5:8	RECORD LENGTH: ASCII or Binary	long	27 lines or 128 bytes
3	9:12	TOTAL RECORDS REFERENCED	long	0
4	13:14	type of data 0 (not supported) 1 = time 2 = spectrum 3 (not supported) 4 = frequency response 5 = cross spectrum 6:9 (not supported) 10 = coherence 11:15 (not supported) 16 = power spectral density 17 = user math 18 = null	e-short	0:18
†5	15:16	x-axis domain 1 = frequency 2 = time 3 (not supported)	e-short	1:3
6	17:18	zoom mode (0=baseband, 1=zoom)	short	0:1
7	19:26	center frequency	double	0:115E+3 Hz
8	27:34	frequency span	double	195.3E−3:102.4E+3 Hz
9	35:42	starting frequency time	double	−65.5E+3:115E+3 Hz or −32.8E+3:8.19E+3 s
10	43:50	ending frequency time	double	−65.5E+3:115E+3 Hz or −32.8E+3:8.19E+3 s
†11	51:54	time record block size	long	512, 1024
12	55:58	# of first point with valid data	long	0:1024
13	59:62	# of last point with valid data	long	0:1024
†14	63:64	measurement mode 1 = linear resolution 2 (not supported)	e-short	1:2
15	65:68	time stamp (six-digit integer: hhmmss)	long	000000:235959
16	69:72	date stamp (six-digit integer: mmddyy)	long	010100:123199
17	73:74	averaging type 0 = RMS 1 = vector 2 = peak hold 3 = off	e-short	0:3
18	75:78	# of averages	long	1:99999
19	79:80	averaging state (0=off, 1=on)	short	0:1
20	81:82	exponential averaging state (0=off, 1=on)	short	0:1
21	83:84	# of channels used for the measurement	short	1:2
22	85:86	channel providing this vector's data 0=not channel specific 1 = channel 1 2 = channel 2	short	0:2
23	87	frequency pair used (0=center/span, 1=start/stop)	bool	0:1
24	88	math overflow during measurement (0=no, 1=yes)	bool	0:1
25	89	measurement occurred in real-time (0=no, 1=yes)	bool	0:1
26	90	data before FFT 0 = complex 1 = real	bool	0:1
27	91:98	reference impedance for dBm units	double	1E−3:1E+7 Ω
—	99:128	padding (binary file only)		

†This field is ignored on recall.

Table 4-18. Measurement State Record

ASCII Index	Binary Index	Meaning of Data/ Data	Data Type	Range/ Units
line 1	byte 1:4	RECORD TYPE	long	201326592 (0C000000 when converted to hexadecimal)
2	5:8	RECORD LENGTH: ASCII or Binary	long	33 lines or 256 bytes
3	9:12	TOTAL RECORD REFERENCE COUNT	long	2
4	13:14	# of channels in measurement	short	1:2
5	15:22	center frequency	double	0:115E+3 Hz
6	23:30	frequency span	double	195.3E−3:102.4E+3 Hz
7	31:38	start frequency	double	(range dependent on span) Hz
8	39:46	stop frequency	double	(range dependent on span) Hz
9	47:54	increment for stepping frequencies	double	15.625E−3:51.2E+3 Hz
10	55:62	time length of record	double	3.906E−3:2.048E+3 s
11	63	zoom mode (0=baseband, 1=zoom)	bool	0:1
12	64:65	active frequency pair (0=start/stop, 1=center/span)	e-short	0:1
13	66:67	trigger type: 0 = continuous 1 = external 2 (not supported) 3 = source 4 = internal, channel 1 5 = internal, channel 2 6 = HP-IB	e-short	0:6
14	68:75	trigger level (as a percentage of input range)	double	−100:100 %
15	76	trigger slope (0=negative, 1=positive)	bool	0:1
16	77	source output state (0=off, 1=on)	bool	0:1
17	78:79	source output type: 0 (not supported) 1 = fixed sine 2 = periodic chirp 3 = random noise	e-short	0:3
18	80:87	source level for random noise output	double	0:5 V
19	88:95	source level for periodic chirp output	double	0:5 V
20	96:103	source level for sine wave output	double	0:5 V
21	104:119	source level label	char[16]	V, Vrms, dBVpk, dBVrms
22	120:127	internal to eng. units conversion factor (for source level)	double	−1E+100:1E+100, except 0
23	128:135	frequency of source sine wave	double	0:115E+3 Hz
24	136:137	arming mode (0=automatic arming, 1=manual arming)	e-short	0:1
25	138	averaging state (0=off, 1=on)	bool	0:1
26	139:140	averaging type 0 = RMS 1 = vector 2 = peak hold 3 = off	e-short	0:3
27	141:144	number of averages	long	1:99999
28	145:146	exponential averaging state (0=off, 1=on)	e-short	0:1
29	147	fast averaging state (0=off, 1=on)	bool	0:1
30	148:151	fast averaging update rate	long	1:99999
31	152:159	time-record overlap per average (requested)	double	0:0.99
32	160:167	reference impedance for dBm calculations	double	1E−3:10E+6 Ω
33	168:171	RECORD REFERENCE: # of channel state records to follow	long	2
—	172:256	padding (binary file only)		

Table 4-19. System State Record

ASCII Index	Binary Index	Meaning of Data/ Data	Data Type	Range/ Units
line 1	byte 1:4	RECORD TYPE	long	285212672 (11000000 when converted to hexadecimal)
2	5:8	RECORD LENGTH: ASCII or Binary	long	21 lines or 128 bytes
3	9:12	TOTAL RECORD REFERENCE COUNT	long	0
† 4	13:14	storage device 0 = no storage 1 = external disc (HP-IB) 2 = internal disc 3 = RAM disc	e-short	0:3
5	15	storage coding (0=ascii, 1=binary)	bool	0:1
6	16	auto-calibration state (0=off, 1=on)	bool	0:1
7	17	calibration trace display (0=off, 1=on)	bool	0:1
8	18	beeper state (0=off, 1=on)	bool	0:1
9	19:20	active plotter speed 0 = fast (36 cm/sec) 1 = slow (5 cm/sec) 2 = user-defined	e-short	0:2
10	21:22	user plotter speed	short	1:100 cm/sec
11	23:24	grid pen number	short	0:64
12	25:26	alpha pen number	short	0:64
† 13	27	system controller state 0 = addressable only 1 = system controller	bool	0:1
14	28:29	HP 35660A's bus address	short	0:30
15	30:31	external disc's bus address	short	0:7
16	32:33	external disc's unit number	short	0:15
17	34:35	external disc's volume number	short	0:7
18	36:37	printer's bus address	short	0:30
19	38:39	plotter's bus address	short	0:30
20	40	HP-IB status annunciators (0=off, 1=on)	bool	0:1
21	41:42	mnemonic display state 1 = mnemonic echo 2 = mnemonic scroll 3 = mnemonic display off	e-short	1:3
—	43:128	padding (binary file only)		

† This field is ignored on recall .

Table 4-20. Trace State Record (part 1)

ASCII Index	Binary Index	Meaning of Data/ Data	Data Type	Range/ Units
line 1	byte 1:4	RECORD TYPE	long	251658240 (0F000000 when converted to hexadecimal)
2	5:8	RECORD LENGTH: ASCII or Binary	long	30 lines or 256 bytes
3	9:12	TOTAL RECORD REFERENCE COUNT	long	0
4	13:14	# of the trace this record describes	short	−1, −2,1,2
† 5	15	trace active (0=not active, 1=active)	bool	0:1
6	16:45	user-defined trace title	char[30]	upper- and lower-case alpha, numbers, spaces, and ;:,.+−*^/_\()[]{}
7	46:47	type of coordinates 0 = linear magnitude 1 = logarithmic magnitude 2 = magnitude 3 = phase 4 = real 5 = imaginary 6 = group delay	e-short	0:6
8	48:49	type of data 0 (not supported) 1 = time 2 = spectrum 3 = power spectral density 4 = frequency response 5 = coherence 6 = cross spectrum	e-short	0:6
9	50:51	channel providing this trace's data 0 = not channel specific 1 = channel 1 2 = channel 2	e-short	0:2
10	52:53	# of active math function (0=no function active)	short	0:5
11	54:55	# of active math constant (0=no function active)	short	0:5
12	56:63	group delay aperture	double	0.5, 1, 2, 4, 8, 16 %
13	64	grid display (0=off, 1=on)	bool	0:1
14	65:66	reserved	short	0
15	67	reference level tracking (0=off, 1=on)	bool	0:1
16	68:83	internal units label (for y-axis per division value)	char[16]	V, V^2, Vrms, Vrms^2, dB, deg, rad, V/rtHz, Vrms/rtHz, V^2/Hz, Vrms^2/Hz, s
17	84:99	engineering units label (for y-axis per div.)	char[16]	upper- and lower-case alpha, numbers, spaces, and ;:,.+−*^/_\()[]{}
18	100:107	internal to eng. units conversion factor (for y-axis per div.)	double	−1E+100:1E+100, except 0

† This field is ignored on recall.

Table 4-20. Trace State Record (part 2)

ASCII Index	Binary Index	Meaning of Data/ Data	Data Type	Range/ Units
line 19	108:115	per div. value for y-axis (in internal units)	double	(dependent on per div. units)
20	116:131	internal units label (for y-axis reference values)	char[16]	V, V^2, Vrms, Vrms^2, dB, dBm, dBVpk, dBVrms, deg, rad, V/rtHz, Vrms/rtHz, V^2/Hz, Vrms^2/Hz, dBVpk/Hz, dBVrms/Hz, dBm/Hz, s
21	132:147	engineering units label (for y-axis ref. values)	char[16]	upper- and lower-case alpha, numbers, spaces, and ;:,.+−*^/_\()[]{}
22	148:155	internal to eng. units conversion factor (for y-axis ref. values)	double	−1E+100:1E+100, except 0
23	156:163	reference value for top of y-axis	double	(dependent on y-axis ref. units)
24	164:171	reference value for center of y-axis	double	(dependent on y-axis ref. units)
25	172:179	reference value for bottom of y-axis	double	(dependent on y-axis ref. units)
26	180:181	reference last changed 0 = top 1 = center 2 = bottom	e-short	0:2
27	182	x-axis scaling (0=linear, 1=logarithmic)	bool	0:1
28	183:184	pen number used to plot this trace	short	0:64
29	185:186	line type used to plot this trace 0 = solid 1 = dotted 2 = dashed 3 = user-defined	e-short	0:4
30	187:188	user-defined line type (The number's meaning is plotter dependent.)	short	0:6, −4096
—	189:256	padding (binary file only)		

Table 4-21. Trace-dependent Marker Record

ASCII Index	Binary Index	Meaning of Data/ Data	Data Type	Range/ Units
line 1	byte 1:4	RECORD TYPE	long	335544320 (14000000 when converted to hexadecimal)
2	5:8	RECORD LENGTH: ASCII or Binary	long	29 lines or 128 bytes
3	9:12	TOTAL RECORD REFERENCE COUNT	long	0
4	13:14	# of the trace this record describes	short	1:2
5	15	main marker state (0=off, 1=on)	bool	0:1
6	16	marker coupling (0=off, 1=on)	bool	0:1
7	17:24	main marker's x-axis value	double	(within selected span) Hz or (within selected time record) s
8	25	offset marker state (0=off, 1=on)	bool	0:1
9	26:33	offset marker's x-axis value	double	0:115E+3 Hz or s
10	34:41	offset marker's y-axis value	double	(dependent on vertical units)
11	42	peak tracking (0=off, 1=on)	bool	0:1
12	43:50	marker search target level	double	(dependent on vertical units)
13	51:52	active special marker 0 = special markers off 1 = harmonic marker 2 = band marker 3 = sideband marker	e-short	0:3
14	53:60	fundamental frequency of harmonic marker	double	0:115E+3 Hz
15	61	harmonic calculation selected (0=harmonic power, 1=THD)	bool	0:1
16	62:63	number of harmonics selected	short	0:400
17	64	harmonic results display (0=off, 1=on)	bool	0:1
18	65:72	carrier frequency for sideband marker	double	0:115E+3 Hz
19	73:80	incremental frequency between sidebands	double	0:115E+3 Hz
20	81	sideband power calculation (0=off, 1=on)	bool	0:1
21	82:83	number of sidebands	short	0:200
22	84:91	lowest frequency of band marker	double	0:115E+3 Hz
23	92:99	highest frequency of band marker	double	0:115E+3 Hz
24	100	band power calculation (0=off, 1=on)	bool	0:1
25	101:102	# of active limit table	short	1:8
26	103	limit lines (0=off, 1=on)	bool	0:1
27	104	limit test (0=off, 1=on)	bool	0:1
28	105	limit beeper (0=off, 1=on)	bool	0:1
29	106	calculate data table (0=off, 1=on)	bool	0:1
—	107:128	padding (binary file only)		

Table 4-22 . Vector Record

ASCII Index	Binary Index	Meaning of Data/ Data	Data Type	Range/ Units
line 1	byte 1:4	RECORD TYPE	long	117571584 (07020000 when converted to hexadecimal)
2	5:8	RECORD LENGTH: ASCII or Binary	long	20:1041 lines or 384:8448 bytes
3	9:12	TOTAL RECORDS REFERENCED	long	0
4	13:14	# of points in the record	short	512 if frequency data, 3:1024 if time data
5	15:94	trace label	char[80]	upper- and lower-case alpha, numbers, spaces, and ;:,.+−*^/_\()[]{}
†6	95:96	# of x-axis values per point	short	0
7	97:136	x-axis domain label	char[40]	FREQUENCY, TIME
8	137:152	x-axis unit label	char[16]	Hz, s
9	153:160	x-axis start point	double	−65.5E+3:115E+3 Hz or −32.8E+3:8.19E+3 s
10	161:168	x-axis increment between points	double	2.44E−3:256 Hz or 3.81E−6:4.0 s
11	169:170	# of y-axis values per point 1 when data is real 2 when data is complex	short	1:2
†12	171:210	y-axis domain label	char[40]	REAL, COMPLEX
13	211:226	y-axis unit label Not listed are the many special units that can result from math operations or the application of engineering units. However, such units are valid here.	char[16]	V, V^2, Vrms, Vrms^2, dB, dBm, dBVpk, dBVrms, deg, rad, V/rtHz, Vrms/rtHz, V^2/Hz, Vrms^2/Hz, dBVpk/Hz, dBVrms/Hz, dBm/Hz, s
†14	227:234	y-axis start point	double	0
†15	235:242	y-axis increment between points	double	0
16	243:244	vector data format (for binary transfers) 1=32-bit binary floating point (IEEE 754-1985) 2=64-bit binary floating point (IEEE 754-1985)	e-short	1:2
17	245:248	skip from record start to data start	long	17 lines or 248 bytes
Vector data starts here. The data consists of a number of points that are directly adjacent to each other in the record. Line 4 specifies the number of points (n) in the record. Line 16 specifies whether points are made up of floats or doubles. Points can take one of two forms. If you are transferring real data, a point takes this form:				
18	249:252 or 249:256	y-axis value	float double	−340E+36:340E+36
—	—	points 2 through n		
—	—	padding to multiple of 128 (binary file only)		
If you are transferring complex data, a point takes this form:				
18	249:252 or 249:256	y-axis value (real part)	float double	−340E+36:340E+36
19	253:256 or 257:264	y-axis value (imaginary part)	float double	−340E+36:340E+36
—	—	points 2 through n		
—	—	padding to multiple of 128 (binary file only)		

†This field is ignored on recall.

Chapter 5
Using the HP 35660A's Status Registers

Introduction

The HP 35660A's status registers contain information about various analyzer conditions. The controller can use one of two methods to access this information:

- The direct-read method — reading the analyzer's registers directly
- The SRQ method — using the analyzer's service request (SRQ) process

In the direct-read method, the analyzer has a passive role. It only tells the controller that conditions have changed when the controller asks the right question. In the SRQ method, the analyzer takes a more active role. It tells the controller when there has been a condition change without the controller asking. Either method allows you to monitor one or more conditions.

When you monitor a condition with the direct-read method, you must:

1. Determine which register contains the bit that monitors the condition.
2. Send the unique HP-IB query that reads that register.
3. Examine the bit to see if the condition has changed.

The direct-read method works well if you do not need to know about changes the moment they occur. It does not work well when you must know about condition changes immediately. Your program would need to continuously read the registers at very short intervals. Since this would make the program relatively inefficient, it would be better to use the SRQ method.

When you monitor a condition with the SRQ method, you must:

1. Determine which bit monitors the condition.
2. Determine how that bit reports to the request service (RQS) bit of the Status Byte.
3. Send HP-IB commands to enable the bit that monitors the condition and to enable the summary bits that report the condition to the RQS bit.
4. Enable the controller to respond to service requests.

When the condition changes, the analyzer sets its RQS bit and the HP-IB's SRQ line. Your program determines how the controller responds to the SRQ, but the important point is this: the controller is informed of the change as soon as it occurs. The time the controller would otherwise have used to monitor the condition can now be used to perform other tasks.

Register Reporting Structure

To use the SRQ method, you must understand how changes in analyzer conditions can result in the HP-IB's SRQ line being set. This requires an understanding of the following items:

- The analyzer's register reporting structure
- The types of registers used in a register set
- The commands and conditions that affect each of the analyzer's register sets

This section discusses the register reporting structure. Subsequent sections discuss the other items.

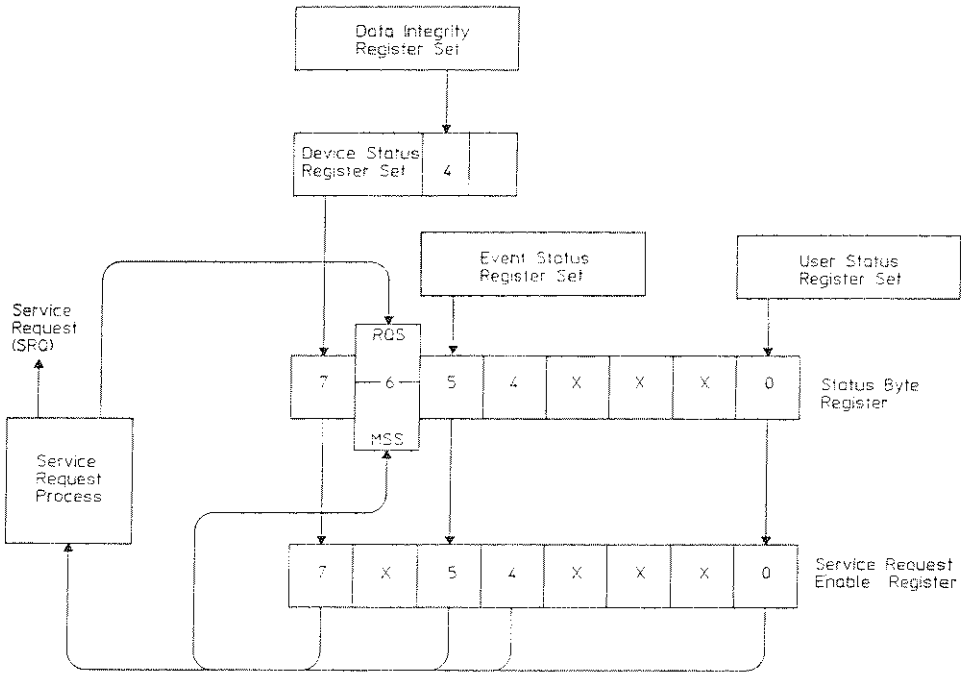


Figure 5-1 Register Reporting Structure

As shown in Figure 5-1, four register sets report to the Status Byte register. The Device Status, Event Status, and User Status register sets all report directly to a particular bit in the Status Byte. The Data Integrity register set reports indirectly to the Status Byte via bit 4 of the Device Status register set.

When a register set causes a Status Byte bit to change from 0 to 1, the analyzer may initiate its service request (SRQ) process. However, the process is only initiated if both of the following conditions are true:

- The corresponding bit of the Service Request enable register is also set to 1
- The analyzer does not have a service request pending (A service request is considered to be pending between the time the analyzer's SRQ process is initiated and the time the controller reads the Status Byte register with a serial poll)

The analyzer's SRQ process sets the HP-IB's SRQ line true and also sets the Status Byte's RQS bit to 1. Both actions are necessary to inform the controller the HP 35660A requires service. Setting the SRQ line only informs the controller that some device on the bus requires service. Setting the RQS bit allows the controller to determine that the HP 35660A, in particular, requires service.

If your program enables the controller to detect and respond to service requests, it should instruct the controller to perform a serial poll when the HP-IB's SRQ line is set true. Each device on the bus returns the contents of its Status Byte register in response to this poll. The device whose RQS bit is set to 1 is the device that requested service.

NOTE When you read the analyzer's Status Byte with a serial poll, the RQS bit is reset to 0. Other bits in the register are not affected.

As implied in Figure 5-1, bit 6 of the Status Byte register serves two functions. Two different methods for reading the register allow you to access the two functions. Reading the register with a serial poll allows you to access the bit's RQS function. See the description of bit 6 later in this chapter for information on accessing the bit's MSS function.

Types of Registers in a Set

The HP 35660A uses four different types of registers in its register sets. The register types are:

1. Condition register
2. Transition registers (one positive, one negative)
3. Event register
4. Enable register

Information Flow in a Register Set

As shown in Figure 5-2, the information flow within a register set starts at the condition register and ends at the register summary bit. You can control the flow of information toward the register summary bit by specifying which bits are set in the transition and enable registers.

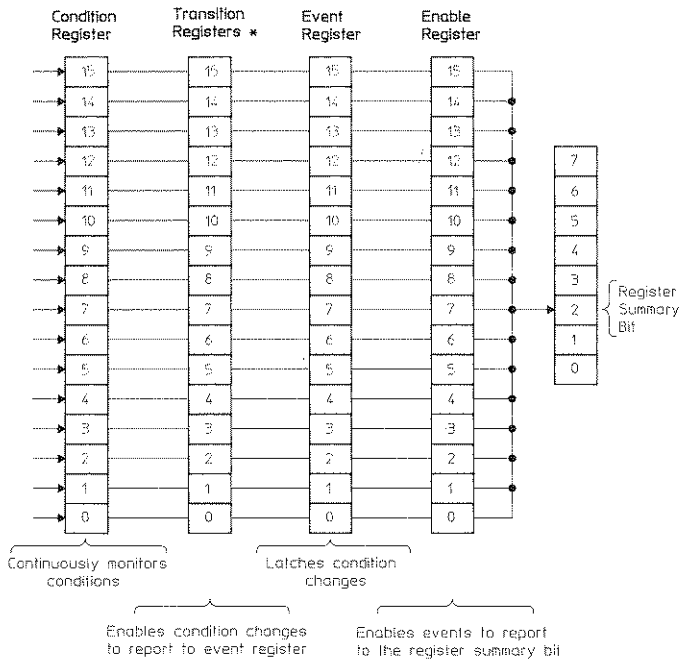


Figure 5-2 Information Flow in a Register Set

The condition register and its two transition registers work together to report condition changes to the event register. Each condition register bit directly monitors a particular analyzer condition. The bit is set to 1 when the condition it monitors becomes true. The bit is reset to 0 when the condition it monitors becomes false. When a condition bit changes from 0 to 1, the change is only reported to the event register if the corresponding bit in the positive transition register is set to 1. When a condition bit changes from 1 to 0, the change is only reported to the event register if the corresponding bit in the negative transition register is set to 1.

The event register and its enable register work together to report latched condition changes to the register summary bit. If an event register bit is reset to 0, the first condition change reported to that bit causes it to be set to 1. Once set, an event bit is no longer affected by condition changes. It remains set until you clear the register. The setting of an event bit is only reported to the register summary bit if the corresponding enable register bit is set to 1.

The register summary bit is only set to 1 when one or more enabled event bits is set to one. It is reset to 0 at all other times.

Special Cases

Two of the analyzer's register sets (Event Status and User Status) only contain an event register and an enable register. In these register sets, event and enable bits serve the same function as in sets that contain all four register types. However, the rule for setting an event bit is slightly modified. Each event bit is assigned to a particular analyzer condition. If the event bit is reset to 0, the first positive transition of the condition (from false to true) causes the event bit to be set to 1. Essentially, the event bit behaves as if a condition bit is reporting to it through a positive transition bit.

One of the analyzer's register sets (the Status Byte register set) contains only a condition register and an enable register. The set consists of the Status Byte register and the Service Request enable register. The Status Byte register is, with the exception of bit 6, a condition register. This means that when the condition monitored by a particular bit is true, that bit is set to 1. When the condition is false, the bit is reset to 0.

Bit 6 of the Status Byte register (when read by the *STB command) serves as the summary bit for the other bits in the register. The Service Request enable register determines which of these other bits will be included in the summary. A Status Byte bit is only included in the summary if the corresponding bit in the Service Request enable register is set to 1.

The HP 35660A's Register Sets

The HP 35660A uses five register sets to keep track of instrument status. The register sets are:

1. The Data Integrity register set – monitors conditions that can effect the validity of your measurement data
2. The Device Status register set – summarizes events in the Data Integrity register set and monitors additional analyzer conditions
3. The User Status register set – detects STAT:USER:PULS commands and key-presses of the instrument's user SRQ softkeys
4. The Event Status register set – detects errors and monitors synchronization conditions
5. The Status Byte register set – summarizes conditions in the other register sets and monitors the analyzer's output queue

The registers sets are summarized graphically in Figure 5-3. They are described in the following sections.

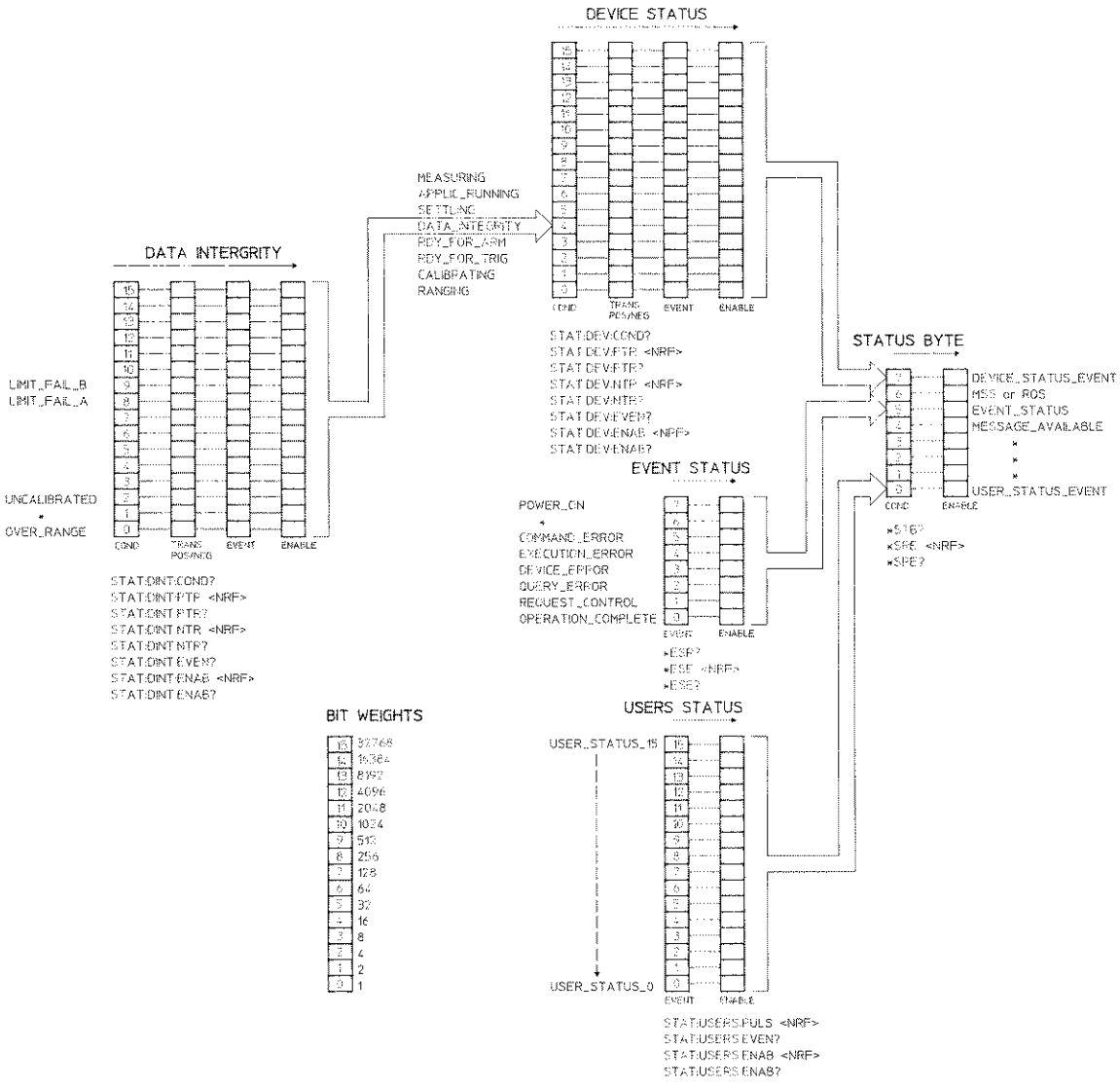


Figure 5-3 HP 35660A Register Summary

Status Byte Register Set

The Status Byte register set contains:

- The Status Byte register (behaves like a condition register for all bits except bit 6)
- The Service Request enable register (for enabling and disabling all bits of the Status Byte register except bit 6)

Power-up States

The state of the Status Byte register at power-up is variable.

The state of the Service Request enable register is saved in nonvolatile memory when you send the SYST:SAVE command. It can be recalled at power-up, depending on the setting of *PSC. If *PSC is 0, the register's state is recalled. If *PSC is 1, all of the register's bits are reset to 0.

Writing to the Registers

You can not write directly to the Status Byte register. Write to the Service Request enable register using the *SRE command.

Reading the Registers

Read the Status Byte register either with the *STB query or with a serial poll. If you read the register with the *STB query, bit 6 serves as the Master Summary Status (MSS) bit. If you read the register with a serial poll, bit 6 serves as the request service (RQS) bit. The other bits' meanings are not affected by the method you use to read the register.

Read the Service Request enable register using the *SRE query.

Clearing the Registers

Clear the Status Byte register by simultaneously clearing all event registers. This is done by sending the *CLS command. You must send the command immediately following a Program Message Terminator (an ASCII line feed character or the HP-IB END message). This ensures that the register's MAV and MSS bits will be cleared. Clear the Service Request enable register by sending *SRE with a value of 0.

Register Summary Bit

Bit 6 of the Status Byte register summarizes all other enabled bits in the Status Byte register, but only if you read the register with the *STB query.

Definition of Bits

Bits 1, 2, and 3 of the Status Byte register set are not used. The other bits in the set are defined in the following sections. Unless otherwise noted, the definitions describe bits in the Status Byte register, not the corresponding bits of the Service Request enable register.

User_Status_Event – bit 0

This bit summarizes all enabled bits of the User Status event register. The User_Status_Event bit is set to 1 when one of the following occurs:

- A bit in the User Status event register changes from 0 to 1 while the corresponding bit of the User Status enable register is set to 1
- A bit in the User Status enable register changes from 0 to 1 while the corresponding bit of the User Status event register is set to 1

To keep the User_Status_Event bit set to 1, there must be at least one case where corresponding bits of the User Status event register and User Status enable register are both set to 1. When there are no such cases, the bit is reset to 0.

Message_Available (MAV) – bit 4

This bit indicates whether or not the analyzer's output queue contains any response messages. The bit is set to 1 when the output queue contains one or more messages. It is reset to 0 when the output queue is empty.

Event_Status (ESB) – bit 5

This bit summarizes all enabled bits of the Event Status register. The Event_Status bit is set to 1 when one of the following occurs:

- A bit in the Event Status register changes from 0 to 1 while the corresponding bit of the Event Status enable register is set to 1
- A bit in the Event Status enable register changes from 0 to 1 while the corresponding bit of the Event Status register is set to 1

To keep the Event_Status bit set to 1, there must be at least one case where corresponding bits of the Event Status register and Event Status enable register are both set to 1. When there are no such cases, the bit is reset to 0.

Request_Service (RQS) or Master_Summary_Status (MSS) – bit 6

This bit is unusual in that it provides different information depending on how the Status Byte register is read. If you read the register with the *STB query, bit 6 summarizes all other enabled bits in the Status Byte register. When bit 6 serves this function, it is known as the Master_Summary_Status (MSS) bit. The MSS bit is reset to 0 when either of the following occurs:

- All enabled Status Byte bits are reset to 0
- All set Status Byte bits are disabled (corresponding bits of the Service Request enable register are reset to 0)

If you read the Status Byte register with a serial poll, bit 6 tells you whether or not the analyzer has requested service. When bit 6 serves this function, it is known as the Request_Service (RQS) bit. The RQS bit is reset to 0 by the same things that reset the MSS bit. But in addition, the RQS bit is reset when the Status Byte register is read by a serial poll. The serial poll does not change the setting of any other bit in the register, not even the MSS portion of bit 6.

Because of the special function of the Status Byte register's bit 6, the setting of the corresponding bit in the Service Request enable register is ignored.

Device_Status_Event – bit 7

This bit summarizes all enabled bits of the Device Status event register. The Device_Status_Event bit is set to 1 when the following occurs:

- A bit in the Device Status event register changes from 0 to 1 while the corresponding bit of the Device Status enable register is set to 1
- A bit in the Device Status enable register changes from 0 to 1 while the corresponding bit of the Device Status event register is set to 1

To keep the Device_Status_Event bit set to 1, there must be at least one case where corresponding bits of the Device Status event register and Device Status enable register are both set to 1. When there are no such cases, the bit is reset to 0.

Event Status Register Set

The Event Status register set contains:

- The Event Status register (an event register)
- The Event Status enable register

Power-up States

Bit 7 of the Event Status register is set to one at power-up. All other bits of that register are reset to 0.

The state of the Event Status enable register is saved in nonvolatile memory when you send the SYST:SAVE command. It can be recalled at power-up, depending on the setting of *PSC. If *PSC is 0, the register's state is recalled. If *PSC is 1, all of the register's bits are reset to 0.

Writing to the Registers

You can not write to the Event Status register. Write to the Event Status enable register using the *ESE command.

Reading the Registers

Read the Event Status register using the *ESR query. Read the Event Status enable register using the *ESE query.

Clearing the Registers

Clear the Event Status register either by reading the register with the *ESR query or by sending the *CLS command. Clear the Event Status enable register by sending *ESE with a value of 0.

Register Summary Bit

Bit 5 of the Status Byte register summarizes all enabled bits of the Event Status register.

Definition of Bits

Bit 6 of the Event Status register set is not used. The other bits in the set are defined in the following sections. Unless otherwise noted, the definitions describe bits in the Event Status register, not the corresponding bits of the Event Status enable register.

Operation_Complete (OPC) -- bit 0

This bit is only set to 1 after the following two events occur in the order listed:

1. You send the *OPC command to the analyzer.
2. The analyzer completes all pending overlapped commands. (For more information, see "Synchronization" in Chapter 2.)

Once set, the bit can only be reset to 0 by clearing the register.

NOTE INIT:STAT STAR and INIT:STAT RUN are considered to be pending overlapped commands whenever bit 7 (Measuring) of the Device Status condition register is set to 1. As a result, the setting of that bit can indirectly affect the setting of the OPC bit. See the description of the Measuring bit to determine how it is set and reset in different measurement situations.

Request_Control (RQC) – bit 1

The analyzer sets this bit to 1 when both of the following are true:

- The analyzer is configured as an addressable-only HP-IB device (See "Controller Capabilities" in Chapter 2.)
- The analyzer is instructed to do something (such as plotting or printing its display) that requires it to take control of the bus

If the controller passes control to the analyzer more than ten seconds before or more than five seconds after this bit is set, the analyzer automatically generates a device error and passes control back. It passes control back to the address specified by the *PCB command.

Once set, the bit can only be reset to 0 by clearing the register.

Query_Error (QYE) – bit 2

This bit is set to 1 when a query error occurs. See Appendix D for a list of conditions that can cause query errors. Once set, the bit can only be reset to 0 by clearing the register.

Device_Error (DDE) – bit 3

This bit is set to 1 when a device-dependent error occurs. See Appendix D for a list of conditions that can cause device-dependent errors. Once set, the bit can only be reset to 0 by clearing the register.

Execution_Error (EXE) – bit 4

This bit is set to 1 when an execution error occurs. See Appendix D for a list of conditions that can cause execution errors. Once set, the bit can only be reset to 0 by clearing the register.

Command_Error (CME) – bit 5

This bit is set to 1 when a command error occurs. See Appendix D for a list of conditions that can cause command errors. Once set, the bit can only be reset to 0 by clearing the register.

Power_On (PON) – bit 7

This bit is set to 1 when you turn the analyzer on. Once set, the bit can only be reset to 0 by clearing the register.

Device Status Register Set

The Device Status register set contains:

- The Device Status condition register
- The Device Status positive transition register
- The Device Status negative transition register
- The Device Status event register
- The Device Status enable register

Power-up States

The state of the Device Status condition register is variable at power-up. All bits of all other Device Status registers are reset to 0.

Writing to the Registers

You can not write to the Device Status condition register or to the Device Status event register. Write to the other Device Status registers using the following commands:

- STAT:DEV:PTR (for the positive transition register)
- STAT:DEV:NTR (for the negative transition register)
- STAT:DEV:ENAB (for the enable register)

Reading the Registers

Read the Device Status registers with the following queries:

- STAT:DEV:COND? (for the condition register)
- STAT:DEV:PTR? (for the positive transition register)
- STAT:DEV:NTR? (for the negative transition register)
- STAT:DEV:EVEN? (for the event register)
- STAT:DEV:ENAB? (for the enable register)

Clearing the Registers

You can not clear the Device Status condition register. Clear the event register either by reading the register with the STAT:DEV:EVEN query or by sending the *CLS command. Clear the transition registers and the enable register by writing to them with a value of 0.

Register Summary Bit

Bit 7 of the Status Byte register summarizes all enabled bits of the Device Status event register.

Definition of Bits

Bits 8 through 15 of the Device Status register set are not used. The other bits in the set are defined in the following sections. Unless otherwise noted, the definitions describe bits of the Device Status condition register.

Ranging – bit 0

This bit is only set to 1 when the analyzer's autoranging routine is enabled and is currently changing the range of one or both input channels. The bit is reset to 0 at all other times. See the description of the INP:RANG:AUTO command for more information on autoranging.

Calibrating – bit 1

This bit is only set to 1 when the analyzer is calibrating. It is reset to 0 at all other times. See the CAL:AUTO command for more information on calibration.

Rdy_for_Trig – bit 2

This bit is only set to one when the analyzer is ready to accept a trigger signal from one of the trigger sources. It is reset to 0 at all other times. If a trigger signal is received before this bit is set, the signal is ignored and the analyzer is not triggered. This bit is most useful when you are using the HP-IB or the external trigger BNC as the trigger source. See commands in the TRIGger subsystem for more information.

Rdy_for_Arm – bit 3

This bit is only set to 1 when both of the following are true:

- Manual arming is turned on (ARM:SOUR HOLD)
- The analyzer is waiting to be armed with the ARM:IMM command

The bit is reset to 0 at all other times. If you send ARM:IMM before this bit is set, the command is ignored and the analyzer is not armed.

Data_Integrity – bit 4

This bit summarizes all enabled bits of the Data Integrity event register. The Data_Integrity bit is set to 1 when one of the following occurs:

- A bit in the Data Integrity event register changes from 0 to 1 while the corresponding bit of the Data Integrity enable register is set to 1
- A bit in the Data Integrity enable register changes from 0 to 1 while the corresponding bit of the Data Integrity event register is set to 1

To keep the Data_Integrity bit set to 1, there must be at least one case where corresponding bits of the Data Integrity event register and Data Integrity enable register are both set to 1. When there are no such cases, the bit is reset to 0.

Settling – bit 5

This bit is only set to 1 when the analyzer is waiting for the digital filter to settle. It is reset to 0 at all other times.

Applic_Running – bit 6

This bit is only set to 1 when an application is running on the analyzer. It is reset to 0 at all other times. If you are using HP Instrument BASIC, this bit is only set when a program is running.

Measuring – bit 7

NOTE INIT:STAT STAR and INIT:STAT RUN are considered pending overlapped commands any time this bit is set to 1. They are considered complete each time this bit changes from 1 to 0. See "Synchronization" in Chapter 2 for more information on overlapped commands.

There are three sets of rules for setting and resetting this bit. There is one set of rules for each of the following measurement situations:

1. Measurement data is unaveraged (AVER:STAT OFF).
2. Measurement data is rms or vector averaged with uniform weighting (AVER:STAT ON, AVER:TYPE RMS or VECT, and AVER:WEIG STAB).
3. Measurement data is averaged with exponential weighting or using the peak hold function (AVER:STAT ON, and AVER:WEIG EXP or AVER:TYPE PEAK).

In the first situation, the bit is set to 1 most of the time a measurement is running. However, each time one or both of the displays are updated with the latest measurement data, the bit is briefly reset to 0. Whenever the measurement is paused, the bit remains at 0 until you restart the measurement. You can pause the measurement by sending INIT:STAT PAUS or by pressing the analyzer's Pause/Cont hardkey.

In the second situation, the bit is set to 1 whenever a measurement is running. The bit is not briefly reset to 0 each time the displays are updated. It is only reset to 0 when the measurement is paused. The analyzer automatically pauses the measurement when the specified number of averages has been collected.

In the third situation, the bit is set to 1 until the first display update after the specified number of averages has been taken. At that point, the bit is briefly reset to 0. Beyond that point, the bit is set to 1 most of the time the measurement is running, but is briefly reset to 0 whenever a display is updated.

Data Integrity Register Set

The Data Integrity register set contains:

- The Data Integrity condition register
- The Data Integrity positive transition register
- The Data Integrity negative transition register
- The Data Integrity event register
- The Data Integrity enable register

Power-up States

The state of the Data Integrity condition register is variable at power-up. All bits of all other Data Integrity registers are reset to 0.

Writing to the Registers

You can not write to the Data Integrity condition register or to the Data Integrity event register. Write to the other Device Status registers using the following commands:

- STAT:DINT:PTR (for the positive transition register)
- STAT:DINT:NTR (for the negative transition register)
- STAT:DINT:ENAB (for the enable register)

Reading the Registers

Read the Data Integrity registers with the following queries:

- STAT:DINT:COND? (for the condition register)
- STAT:DINT:PTR? (for the positive transition register)
- STAT:DINT:NT?R (for the negative transition register)
- STAT:DINT:EVEN? (for the event register)
- STAT:DINT:ENAB? (for the enable register)

Clearing the Registers

You can not clear the Data Integrity condition register. Clear the event register either by reading the register with the STAT:DINT:EVEN query or by sending the *CLS command. Clear the transition registers and the enable register by writing to them with a value of 0.

Register Summary Bit

Bit 4 of the Device Status condition register summarizes all enabled bits of the Data Integrity event register.

Definition of Bits

Bits 1, 3 through 7, and 10 through 15 of the Data Integrity register set are not used. The other bits in the set are defined in the following sections. Unless otherwise noted, the definitions describe bits of the Data Integrity condition register.

Over_Range – bit 0

This bit is only set to 1 when the amplitude of a signal entering an input channel exceeds the current range setting of that channel. The bit is set when this is true for either one or both channels. It is reset to 0 at all other times.

NOTE When the analyzer is in the one-channel measurement mode (CONF:TYPE SPEC), channel 2 does not set the Over_Range bit unless the signal entering that channel exceeds 27 dBVrms. This is because channel 2 is automatically set to its highest range (27 dBVrms) when the one-channel mode is selected.

Uncalibrated – bit 2

This bit is only set to 1 when there are no calibration constants available to correct the measurement data. It is reset to 0 at all other times. Calibration constants are always available unless one of the following is true:

- The analyzer has been unable to complete a calibration
- The analyzer has not been calibrated since the CLEAR CAL CONSTANTS softkey was pressed
- The analyzer has not been calibrated since the CAL:CLE command was sent

Limit_Fail_A – bit 8

This bit is always reset to 0 when limit testing is turned off for display A. When limit testing is turned on, the following rules determine when the bit is set and reset:

The bit is briefly reset to 0 each time measurement data is evaluated against the specified limits. If the data passes the limit test, the bit remains at 0 until the next evaluation. If the data fails the limit test, the bit is set to 1 until the next evaluation.

Measurement data is evaluated each time the display is updated (this occurs even if display blanking is on).

Limit_Fail_B – bit 9

The rules for setting and resetting this bit are the same as the rules for setting Limit_Fail_A, except that this bit monitors limit tests on display B.

User Status Register Set

The User Status register set contains:

- The User Status event register
- The User Status enable register

Power-up States

All bits of the two User Status registers are reset to 0 at power-up.

Writing to the Registers

Write to the User Status event register using the USER:STAT:PULS command. (Pressing one of the analyzer's User SRQ softkeys also writes to the register.) Write to the enable register using the STAT:USER:ENAB command.

Reading the Registers

Read the User Status event register using the STAT:USER:EVEN query. Read the User Status enable register using the STAT:USER:ENAB query.

Clearing the Registers

Clear the User Status event register either by reading the register with the STAT:USER:EVEN query or by sending the *CLS command. Clear the User Status enable register by sending STAT:USER:ENAB with a value of 0.

Register Summary Bit

Bit 0 of the Status Byte register summarizes all enabled bits of the User Status event register.

Definition of Bits

The bits in the User Status register set are User_Status_0 through User_Status_15. Bits 0 through 9 of the event register can be set either by pressing one of the analyzer's User Status softkeys or by sending a value with the STAT:USER:PULS command. Bits 10 through 15 of the event register can only be set with the STAT:USER:PULS command. Once set, an event register bit can only be reset to 0 by clearing the register.

Chapter 6
Programming Examples

Introduction

This chapter contains listings of 11 example programs stored on the HP 35660A Getting Started disc. The listings, which are organized alphabetically by filename, demonstrate many important programming concepts, including:

- Measurement synchronization
- Passing control
- Transferring data
- Generating service requests (SRQs)

All of the programs were written in HP BASIC 5.0 for use on an HP Series 200 computer. However, numerous comments make it possible for you to adapt the programs to other languages and computers.

Example 1.

```
10    !BASIC Program: DATATBL -  Load and Read a Data Table
20    !
30    !This program is divided into three parts.  The first part
40    !accepts x-values for the data table from the keyboard.  The
50    !second part sends the data table to the instrument.  Part
60    !three initiates a measurement, waits for the measurement to
70    !complete, and then reads and displays the x and y-values.
80    !NOTE: Use HP35660A front panel keys to view data table.
90    !
100   Scode=7                                    !Interface select code
110   Address=11                                 !Address for HP 35660A
120   Dsa=Scode*100+Address
130   DIM Data_tbl(1:20,1)                       !20 x 2 data table array
140   INTEGER X,Y
150   X=0                                        !X index for array (always = 0)
160   Y=1                                        !Y index for array (always = 1)
170   !
180   ASSIGN @Dsa TO Dsa;FORMAT ON               !Use for ASCII data
190   ASSIGN @Dsa_off TO Dsa;FORMAT OFF          !Use for binary data
200   !
220   CLEAR SCREEN
230   OUTPUT @Dsa;"MARK:DTAB:HEAD:AFOR FP64"  !Set up for binary transfers
240   !
250 Generate_table:                              !Enter data from keyboard
260   INPUT "Number of points in table?",Num_points
270   FOR I=1 TO Num_points
280     DISP "Enter x-axis value for data point #";I;
290     INPUT New_point
300     Data_tbl(I,X)=New_point      !Save point into point array
310     PRINT I,New_point            !Display the new point
320   NEXT I
330   !
340 Send_table:     !Send table to instrument.  Only X values can be sent
350   !Build a header
360   Block_count=Num_points*8               !Number of bytes in data block
370   Byte_count=LEN(VAL$(Block_count))      !Number of digits in Block_count
380   !
390   OUTPUT @Dsa;"MARK:A:DTABL:DATA";       !Start sending the data
400   !First send the data block header
410   OUTPUT @Dsa USING "#,A,D,"&VAL$(Byte_count)&"D";"#",Byte_count,Block_count
420   !
430   FOR I=1 TO Num_points                  !Start sending the data block
440     OUTPUT @Dsa_off;Data_tbl(I,X);       !OUTPUT each data point
450   NEXT I
460   OUTPUT @Dsa;CHR$(10)                   !Output a LF character (EOL)
470   !
480 Read_table:   !Read and display the data table after a completed measurement
490   CLEAR SCREEN
500   DISP "Starting the measurement...";
510   OUTPUT @Dsa;"MARK:A:DTAB:STAT ON"      !Turn the calculation on
520   OUTPUT @Dsa;"INIT:STAT STAR;*WAI"      !Start the measurement
530   !
540   OUTPUT @Dsa;"MARK:A:DTABL:DATA?"       !Read table after measurement done
550   ENTER @Dsa USING "#,A,D";A$,Byte_count  !first byte of block header
560   ENTER @Dsa USING "#,"&VAL$(Byte_count)&"D";Block_count
570   DISP "DONE"
580   !
590   PRINT TABXY(4,1);"X";TABXY(11,1);"RESULTS";TABXY(24,1);"Y"
600   FOR I=1 TO (Block_count/8)/2       !FOR I=2 TO 'Pairs of FP numbers'
610     ENTER @Dsa_off;Data_tbl(I,X),Data_tbl(I,Y)              !Read X,Y
620     PRINT TAB(1);Data_tbl(I,X);TAB(20);Data_tbl(I,Y)        !PRINT X,Y
630   NEXT I
640   ENTER @Dsa USING "A";A$                !Read LF character at end of block
650   !
660   END
```

Example 2.

```
10    ! BASIC Program:  DUMPTRACE - Reading trace data
20    ! Read the coordinate transformed data from Trace A
30    !
40    Scode=7                              !Interface select code
50    Address=11                           !Address of HP 35660A
60    Dsa=Scode*100+Address
70    ASSIGN @Dsa_off TO Dsa;FORMAT OFF
80    CLEAR SCREEN
90    !
100   DIM Trace1(1:401)                    !Most displays return 401 points
110   DIM Trace2(1:1024)                   !Time display can return 1024 points
120   DIM Name$[80]                        !Trace title
130   INTEGER Byte_count1                  !Number of bytes in Block_count
140   INTEGER Block_count                  !Number of bytes in data block
150   !
160   OUTPUT Dsa;"DISP:HEAD:AFOR FP64"     !Set txfer format to binary
170   OUTPUT Dsa;"DISP:A:DATA?"            !Request the display data block
180   !
190   ! Read the data block header.  The header consists of a '#' followed
200   ! by: a single digit ASCII number,a second ASCII number, the data block,
210   ! and a terminator.  The single digit indicates how many bytes make up
220   ! the second number.  The second number indicates how many bytes are
230   ! in the data block.  For FP64 there are 8 bytes for every data point.
240   !     Example (401 points):  #43208<3208 bytes><LF>
250   !
260   ENTER Dsa USING "%,A,D";A$,Byte_count1
270   ENTER Dsa USING "%,"&VAL$(Byte_count1)&"D";Block_count
280   Num_points=Block_count/8
290   SELECT Num_points                     !Select the right sized array
300   CASE 401
310     ENTER @Dsa_off;Trace1(*)            !Read the 3208 bytes into 401 points
320     Max_val=MAX(Trace1(*))              !Find the peak value
330   CASE 1024
340     ENTER @Dsa_off;Trace2(*)            !Read the 8192 bytes into 1024 points
350     Max_val=MAX(Trace2(*))              !Find the peak value
360   CASE ELSE
370     DISP "This program can't handle block sizes of";Num_points
380     CLEAR Dsa      !Clear unread data from HP35660A output buffer
390     GOTO End
400   END SELECT
410   !
420   ENTER Dsa USING "A";A$           !Read termination character
430   PRINT "Successfully read ";Num_points;"Data points"
440   !
450   OUTPUT Dsa;"DISP:A:HEAD:NAME?"   !Read the Trace title
460   ENTER Dsa;Name$
470   OUTPUT Dsa;"DISP:A:HEAD:XOR?"    !Read the x-axis starting value
480   ENTER Dsa;X_start
490   OUTPUT Dsa;"DISP:A:HEAD:XINC?"   !Read the x-axis increment/bin
500   ENTER Dsa;Xinc
510   OUTPUT Dsa;"DISP:A:HEAD:XUN?"    !Read the x-axis units
520   ENTER Dsa;X_units$
530   OUTPUT Dsa;"DISP:A:HEAD:YUN?"    !Read the y-axis units
540   ENTER Dsa;Y_units$
550   X_stop=(Num_points-1)*Xinc+X_start   !Calculate the x-axis ending value
560   !
570   PRINT "Trace title is: ";Name$
580   PRINT USING "K,6D.5D,X,K";"Start:",X_start,FNStrip$(X_units$)
590   PRINT USING "K,6D.5D,X,K";"Stop :",X_stop,FNStrip$(X_units$)
600   PRINT USING "K,X,6D.5D,X,K";"Maximum value is:",Max_val,FNStrip$(Y_units$)
610 End:LOCAL Dsa                      !Return the 35660A to Local
620   END
630   DEF FNStrip$(A$)                 !Strip quotes from around string
640     RETURN A$[2,LEN(A$)-1]         !Return all but first and last characters
650   FNEND
```

Example 3.

```
10    !BASIC Program:  EXPAND - Read/Write complex trace data
20    !
30    !Designed to demonstrate how to read and write a complex trace,
40    !this program will read a trace, expand it around the current
50    !marker position and then send the trace back to the HP35660A.
60    !The amount of expansion is selected using softkeys.   To keep
70    !the program as simple as possible, provisions for handling real
80    !time data (1024 x 1 points) were not incorporated.
90    !
100   Zoom=0          !Current zoom factor, 0 indicates no valid data
110   Scode=7         !Interface select code
120   Address=11      !Address for HP 35660A
130   Dsa=Scode*100+Address
140   !
150   DIM Trace_in(511,1)     !Array for trace read  512x2 points
160   DIM Trace_out(511,1)    !Array for expanded trace 401x2 points
170   !
180   ASSIN @Dsa TO Dsa;FORMAT ON                 !Use for ASCII
190   ASSIGN @Dsa_off TO Dsa;FORMAT OFF           !Use for binary
200   !
210   !Set up softkeys
220   KBD CMODE ON
230   ON KEY 0 LABEL "READ TRACE" GOSUB Read_trace
240   ON KEY 1 LABEL "RESTORE TRACE" GOSUB Restore_trace
250   ON KEY 2 LABEL "ZOOM 2X" GOSUB Expand_by_two
260   ON KEY 3 LABEL "ZOOM ?X" GOSUB Expand_by_arb
270   ON KEY 4 LABEL "AUTO SCALE" GOSUB Auto_scale
280   FOR I=5 TO 9                                !Clear unused softkeys
290     ON KEY I LABEL "" GOTO Wait
300   NEXT I
310   !
320 Wait:LOOP                                     !Wait here for key press
330   END LOOP
340  !
350 Read_trace:             !Subroutine to read a 512 x 2  point trace
360   DISP "Reading trace data..."
370   INTEGER Byte_count,Block_count
380   OUTPUT @Dsa;"TRAC:HEAD:AFOR FP64"           !Set data transfer to binary
390   OUTPUT @Dsa;"TRAC:DATA?"                    !Request data
400   ENTER @Dsa USING "%,A,D";A$,Byte_count      !Read block header
410   ENTER @Dsa USING "%,"&VAL$(Byte_count)&"D";Block_count
420   ENTER @Dsa_off;Trace_in(*)                  !Read the data block
430   ENTER @Dsa USING "A";A$                     !Read LF character
440   !
450   OUTPUT @Dsa;"TRAC:HEAD:XOR?"                !Read x-axis origin
460   ENTER @Dsa;X_origin
470   OUTPUT @Dsa;"TRAC:HEAD:XINC?"               !Read x-axis increment/bin
480   ENTER @Dsa;X_incr
490   OUTPUT @Dsa;"TRAC:HEAD:XUN?"                !Read x-axis units
500   ENTER @Dsa;Xunit$
510   OUTPUT @Dsa;"TRAC:HEAD:YPO?"                !Read y-points/bin
520   ENTER @Dsa;Ypo
530   IF Ypo=2 THEN                               !Check for complex data
540     Zoom=1                                    !Allow zooming, current zoom=1
550     OUTPUT @Dsa;"MARK:A:STAT ON"              !Make sure marker is on
560     DISP "Move marker to center of expansion before zooming."
570   ELSE
580     BEEP
590     Zoom=0
600     DISP "Can't expand real time data -- only complex."
610   END IF
620   LOCAL Dsa                                   !Return the analyzer to LOCAL
630   RETURN
640     !
650 Restore_trace:  !RESTORE TRACE softkey pressed.  Send original trace back
660   IF Zoom=0 THEN GOTO No_data                 !No valid data, abort
```

Example 3 (continued.)

```
670   Zoom=1                                          !New zoom value
680   GOTO Do_it                                      !Do the zoom
690   !
700 Expand_by_two:   !ZOOM 2X softkey pressed.  Change zoom by two.
710   IF Zoom=0 THEN GOTO No_data                     !No valid data, abort
720   Zoom=2*Zoom                                     !New zoom value
730   GOTO Do_it                                      !Do the zoom
740   !
750 Expand_by_arb:   !ZOOM ?X softkey was pressed, enter zoom value
760   IF Zoom=0 THEN GOTO No_data                     !No valid data, abort
770   !
780   REPEAT
790     INPUT "Enter zoom factor: ",Zoom_entry
800   UNTIL Zoom_entry>0                              !Don't allow negative numbers
810   Zoom=Zoom_entry*Zoom                            !New zoom=entry*current zoom
820   GOTO Do_it                                      !Do the zoom
830   !
840 Do_it:                                            !Do the zoom
850   IF Zoom<>1 THEN GOSUB Expand                    !Calculate new trace
860   GOSUB Read_out                                  !Send new trace to analyzer
870   LOCAL Dsa                                       !Return the analyzer to LOCAL
880   RETURN                                          !Return from ON KEY
890     !
900 No_data:BEEP                                      !Haven't got valid data yet
910   DISP "READ TRACE first"
920   RETURN                                          !Return from ON KEY
930       !
940 Auto_scale:       !Send an auto-scale instruction to the analyzer
950   OUTPUT @Dsa;"DISP:A:SCAL:AUTO:SING"
960   LOCAL Dsa
970   RETURN                                          !Return from ON KEY
980                   !
990 Expand: !
1000  INTEGER J,Index_in,Index_out,I
1010  !
1020  DISP "Expanding Trace ..."
1030  OUTPUT @Dsa;"MARK:X?"                            !Read marker frequency
1040  ENTER @Dsa;Marker_x
1050  !Calculate bin number for unzoomed trace
1060  Marker_bin=(Marker_x-X_origin)/X_incr
1070  !
1080  I=0
1090  IF Xunit$[2,2]="S" THEN              !Check for time trace (units=Seconds)
1100    Top_bin=511                        !Use all 512 points of time trace
1110    Center_bin=256
1120  ELSE
1130    Top_bin=400                        !Use first 401 points of freq trace
1140    Center_bin=200
1150  END IF
1160  Index_out=Center_bin                 !Start in the middle of Trace_out()
1170  WHILE Index_out<=Top_bin             !and work up to the top
1180    Index_in=Marker_bin+(I DIV Zoom)   !Calculate which bin to get data from
1190    IF Index_in<=Top_bin THEN          !Use data from Trace_in()
1200      Trace_out(Index_out,0)=Trace_in(Index_in,0)    !Real part
1210      Trace_out(Index_out,1)=Trace_in(Index_in,1)    !Imaginary part
1220    ELSE                               !Ran out of data, use 0
1230      Trace_out(Index_out,0)=0         !Real Part
1240      Trace_out(Index_out,1)=0         !Imaginary Part
1250    END IF
1260    Index_out=Index_out+1              !Next bin up
1270    I=I+1
1280  END WHILE
1290  I=0
1300  Index_out=Center_bin-1               !Start at middle-1 and work down
1310  WHILE Index_out>=0
1320    Index_in=Marker_bin-(I DIV Zoom)-1
```

Example 3 (continued)

```
1330    IF Index_in>=0 THEN                !Use data from Trace_in()
1340      Trace_out(Index_out,0)=Trace_in(Index_in,0)    !Real part
1350      Trace_out(Index_out,1)=Trace_in(Index_in,1)    !Imaginary part
1360    ELSE                               !Ran out of data, use 0
1370      Trace_out(Index_out,0)=0
1380      Trace_out(Index_out,1)=0
1390    END IF
1400    Index_out=Index_out-1              !Next bin down
1410    I=I+1
1420  END WHILE
1430  RETURN                               !Return to Do_it
1440    !
1450 Read_out:  !Subroutine to send trace data back to the analyzer
1460  DISP "Current zoom factor = ";Zoom          !Show the current zoom
1470    !
1480  IF Zoom=1 THEN                              !Restore original x-axis scale
1490    OUTPUT @Dsa;"TRAC:HEAD:XOR ";X_origin     !Set x-axis origin
1500    OUTPUT @Dsa;"TRAC:HEAD:XINC ";X_incr      !Set x-axis increment/bin
1510  ELSE                                        !Set new x-axis scale
1520    OUTPUT @Dsa;"TRAC:HEAD:XOR ";Marker_x-Center_bin*X_incr/Zoom
1530    OUTPUT @Dsa;"TRAC:HEAD:XINC ";X_incr/Zoom
1540  END IF
1550  OUTPUT @Dsa;"TRAC:HEAD:YPO ";Ypo            !Set y-points/bin
1560    !
1570  OUTPUT @Dsa;"TRAC:DATA";                    !Data's on its way
1580  IF Zoom=1 THEN        !Send the original data back
1590    OUTPUT @Dsa USING "#,A,D,4D";"#",4,512*2*8  !Send a header '#48192'
1600    OUTPUT @Dsa_off;Trace_in(*),CHR$(10)
1610  ELSE                  !Send the expanded data back
1620    OUTPUT @Dsa USING "#,A,D,4D";"#",4,512*2*8  !Send a header '#48192'
1630    OUTPUT @Dsa_off;Trace_out(*),CHR$(10)
1640  END IF
1650  RETURN
1660 Prog_end:END
```

Example 4.

```
10    !BASIC Program:  LIMITTBL - Downloading a limit table
20    !
30    !This program creates a new limit table from information stored
40    !in DATA statements and downloads that table into the HP 35660A
50    !
60    Scode=7                                  !Interface select code
70    Address=11                               !Address for HP 35660A
80    Dsa=Scode*100+Address
90    ASSIGN @Dsa TO Dsa;FORMAT ON             !Use this IO path for ASCII data
100   ASSIGN @Dsa_off TO Dsa;FORMAT OFF        !Use this IO path for binary data
110   !
120   DISP "Presetting the HP35660A..."
130   OUTPUT @Dsa;"*RST"                       !Preset the HP35660A
140   OUTPUT @Dsa;"DISP:A:SCAL:STOP -51 DBVRMS"   !Set display scale
150   OUTPUT @Dsa;"DISP:A:SCAL:DIV 10 DB;*OPC?"
160   ENTER @Dsa;Opc                           !Wait here until setup complete
170   !
180   DIM Table(1:20,1:5)                      !20 segments, 5 pts/seg
190   OUTPUT @Dsa;"LIMIT1:TABL:HEAD:AFOR FP64"    !Set up for binary transfer
200   !
210   DISP "Generating limit table...";
220   !First number in DATA is the number of segments defined
230   READ Segment_count
240   !
250   FOR I=1 TO Segment_count                  !Read data for each segment
260     FOR J=1 TO 5                            !Read all five segment parameters
270       READ Table(I,J)
280     NEXT J
290   NEXT I
300  !
310 Output_table:          !Send the data in Table() to the 35660A as LIM1
320   Block_count=Segment_count*5*8             !8 bytes/number, 5 numbers/seg
330   Byte_count=LEN(VAL$(Block_count))         !Number of digits in Block_count
340     !
350   OUTPUT @Dsa;"LIM1:TABL:DATA";             !Tell HP35660A that data is coming
360   !Send the data header "#<byte_count><block count>"
370   OUTPUT @Dsa USING "#,A,D,"&VAL$(Byte_count)&"D";"#",Byte_count,Block_count
380   !
390   FOR I=1 TO Segment_count
400     FOR J=1 TO 5
410       OUTPUT @Dsa_off;Table(I,J); !Send data in 64 bit floating point format
420     NEXT J
430   NEXT I
440   OUTPUT @Dsa;CHR$(10)                      !OUTPUT a LF character to end block
450   OUTPUT @Dsa;"DISP:A:LIM 1"                !Associate table 1 with trace A
460   OUTPUT @Dsa;"DISP:A:LIM:LINE ON"          !Turn the limit lines on
470   DISP "DONE"
480   STOP
490   !
500   !DATA for Limit lines
510   !Total number of segments in table
520   DATA 6
530   !
540   !The data (one segment per line) is arranged as:
550   !   DATA x-start, x-stop, y-start, y-stop, y-flag
560   !NOTE: Values assume units for trace A (e.g. Hz and dBVrms)
570   DATA 11000, 13600, -100, -60, 0
580   DATA 12300, 21300, -80, -80, 0
590   DATA 20000, 22600, -100,-60, 0
600   DATA 30000, 32600, -100, -60, 0
610   DATA 25800, 34200, -100, -100, 0
620   DATA 28400, 36800, -60,-60, 0
630   END
```

Example 5.

```
10    !BASIC program: MEAS_SYNC - Measurement synchronization
20    !
30    !This program demonstrates how to use the MEASURING bit in
40    !the DEVICE STATUS register to interrupt a program when a
50    !measurement is complete.  The program will read and display
60    !the marker value with every trace update.
70    !
80    Scode=7                          !Interface select code
90    Address=11                       !Address for HP 35660A
100   Dsa=Scode*100+Address
110   !
120   DISP "Presetting the HP35660A..."
130   OUTPUT Dsa;"*RST; *OPC?"         !Preset the HP35660A
140   ENTER Dsa;Opc                    !Wait here until preset complete
150   OUTPUT Dsa;"SWE:TIME 2"          !Set record length to 2 seconds
160   OUTPUT Dsa;"AVER ON"             !Turn averaging on
170   OUTPUT Dsa;"AVER:TYPE RMS;WEIG EXP"  !Set averaging type to EXPON
180   OUTPUT Dsa;"AVER:COUN 2"         !Set number of averages
190   !
200   OUTPUT Dsa;"*CLS"                !Clear STATUS BYTE register
210   OUTPUT Dsa;"STAT:DEV:NTR 128"    !Program DEVICE STATUS NTR register
220   OUTPUT Dsa;"STAT:DEV:ENAB 128"   !Program DEVICE STATUS ENAB register
230   OUTPUT Dsa;"*SRE 128"            !Program STATUS BYTE ENAB register
240   ON INTR Scode,2 GOSUB Srq_handler   !Set up interrupt branching
250   ENABLE INTR Scode;2                 !Allow interrupt on SRQ
260   !
270   OUTPUT Dsa;"INIT:STAT STAR"      !Start the measurement
280   DISP "Waiting for first measurement to complete..."
290 Idle:                              !Wait here for interrupt
300   GOTO Idle
310   !
320 Srq_handler:   !Got an SRQ
330   Stb=SPOLL(Dsa)                   !Read STATUS BYTE and clear SRQ
340   OUTPUT Dsa;"STAT:DEV:EVEN?"      !Read and clear DEVICE STATUS EVENT reg.
350   ENTER Dsa;Dse
360     !
370   OUTPUT Dsa;"MARK:A:AMPL?"        !Read the marker amplitude
380   ENTER Dsa;Mark_ampl
390   DISP "Marker amplitude: ";Mark_ampl
400   !
410   ENABLE INTR Scode                !Re-enable the interrupts
420   RETURN
430   !
440   END
```

Example 6.

```
10    !HP-IB program: OPC_SYNC - Measurement synchronization
20    !
30    !This program demonstrates the how to use the *OPC command to
40    !allow an SRQ to interrupt program execution.  *OPC will set
50    !the OPERATION_COMPLETE bit in the EVENT STATUS register
60    !when all pending HP-IB commands have finished.  With the proper
70    !register masks, this will generate a service request.
90    !
100   Scode=7                              !Interface select code
110   Address=11                           !Address for HP 35660A
120   Dsa=Scode*100+Address
130   !
140   OUTPUT Dsa;"SWE:TIME 8"              !Set record length to 8 seconds
150   OUTPUT Dsa;"*CLS"                    !Clear the STATUS BYTE register
160   OUTPUT Dsa;"*ESE 1"                  !Program the EVENT STATUS ENABLE reg.
170   OUTPUT Dsa;"*SRE 32"                 !Program the STATUS BYTE ENABLE reg.
180   !
190   ON INTR Scode,2 GOTO Srq_handler     !Set up interrupt branching
200   ENABLE INTR Scode;2                  !Allow SRQ to generate an interrupt
210   !
220   OUTPUT Dsa;"INIT:STAT STAR"          !Start the measurement
230   OUTPUT Dsa;"*OPC"                    !Generate SRQ when all commands have
240                                        !finished.
250   Start_time=TIMEDATE
260   LOOP                                 !Do something useful while waiting
270     DISP USING "14A, 2D.D";"Elapsed time :",TIMEDATE-Start_time
280     WAIT .1
290   END LOOP
300   !
310 Srq_handler:   !Got an SRQ
320     Stb=SPOLL(Dsa)                     !Read STATUS BYTE and clear SRQ
330     BEEP
340     OUTPUT Dsa;"*ESR?"                 !Read and clear EVENT STATUS reg.
350     ENTER Dsa;Esr
360     DISP "Got the SRQ! SPOLL returns:";Stb;"   ESR returns:";Esr
370   END
```

Example 7.

```
10    !BASIC program: OPCQ_SYNC -  Measurement synchronization
20    !
30    ! This program demonstrates how to use the *OPC? HP-IB command
40    ! to hang the bus on a query before continuing on with the
50    ! BASIC program.  After all pending HP-IB commands have finished,
60    ! the HP35660A will return a '1' in response to *OPC?.
70    !
80    Scode=7
90    Dsa=Scode*100+11
100   !
110   DISP "Presetting the HP35660A..."
120   OUTPUT Dsa;"*RST"                    !Preset the HP35660A
130   OUTPUT Dsa;"*OPC?"                   !Pause on ENTER statement until
140   ENTER Dsa;Opc                        !'*RST' command has finished
150   !
160   OUTPUT Dsa;"SWE:TIME 8"              !Set record length to 8 seconds
170   DISP "Measurement started ..."
180   OUTPUT Dsa;"INIT:STAT STAR"          !Start the measurement
190   OUTPUT Dsa;"*OPC?"                   !Pause until all pending HP-IB commands
200   ENTER Dsa;Opc                        !have finished.
210   BEEP
220   DISP "Measurement done"
230   END
```

Example 8.

```
10    ! BASIC Program: PASSCNTL - Passing control to HP35660A
20    !
30    ! This program instructs the HP35660A perform a screen dump to a
40    ! printer and generate a service request when done.  Control is
50    ! passed to the HP35660A when the print command is issued and
60    ! automatically passed back when the instrument no longer needs it.
70    !
80    Scode=7                                 !Interface select code
90    Address=11                              !Address for HP35660A
100   Dsa=Scode*100+Address
110   OUTPUT Dsa;"*CLS"                       !Clear the STATUS BYTE register
120   !
130   ! Program the instrument to generate SRQ on OPERATION_COMPLETE. This
140   ! requires programming the STATUS BYTE and EVENT STATUS enable regs.
150   OUTPUT Dsa;"*ESE 1"                     !Bit 1 = OPERATION_COMPLETE
160   OUTPUT Dsa;"*SRE 32"                    !Bit 5 = EVENT_STATUS
170   OUTPUT Dsa;"GPIB:LEDS ON"               !Turn on HP-IB status LED's
180   OUTPUT Dsa;"*PCB 21"                    !Set up Pass control back address
190   !
200   ON INTR Scode GOTO Srq_handler          !Set up interrupt branching
210   ENABLE INTR Scode;2                     !Enable interrupt on SRQ
220   !
230   DISP "HP35660A Printing screen..."
240   OUTPUT Dsa;"PRIN:DUMP:SCR"              !Instruct analyzer to print the screen
250   OUTPUT Dsa;"*OPC"                       !Set OPC bit when everythings complete
260   PASS CONTROL Dsa                        !Give control of the bus to the 35660A
270   !
280 Wait_here:                                !Wait for OPC to generate an interrupt
290   GOTO Wait_here
300 !
310 Srq_handler:                              !If there's an interrupt, then
320                                           !Control was passed back
330   IF BINAND(SPOLL(Dsa),64) THEN           !HP35660A is requesting service
340     BEEP
350     DISP "HP35660A Done Printing"
360   ELSE                                    !It wasn't the HP35660A
370     DISP "UNKNOWN SRQ"
380   END IF
390   END
```

Example 9.

```
10    ! HP-IB Program: USERSRQ -  Responding to USER SRQ's
20    !
30    ! This program demonstrates how user generated service
40    ! requests can be used to interrupt a program.
50    !
60    Scode=7                              !Interface select code
70    Address=11                           !Address of HP35660A
80    Dsa=Scode*100+Address
90    INTEGER User_status_reg              !16 bit integer
100   !
110   OUTPUT Dsa;"*CLS"                    !Clear the STATUS BYTE register
120   !
130   !Set USER STATUS ENABLE register to all 1's.
140   OUTPUT Dsa;"STAT:USER:ENAB 65535"  ! 65535 = 1+2+4+8+....+2^15
150   !
160   !Set STATUS BYTE ENABLE register for SRQ on USER_STATUS_EVENT only
170   OUTPUT Dsa;"*SRE 1"
180   LOCAL Dsa                            !Put the instrument in LOCAL mode
190   !
200   !Instrument is set up;  Enable interrupts to detect an SRQ
210   ON INTR Scode GOSUB Srq_handler      !Set up interrupt branching
220   ENABLE INTR Scode;2                  !Enable interrupt on SRQ
230   !
240   CLEAR SCREEN                         !Clear the text
250 Wait:DISP "On the HP35660A, Press [Local HP-IB] <USER SRQ> <SRQx>"
260   GOTO Wait                            !Wait for SRQ to occur
270   !
280 Srq_handler:   !
290   IF BINAND(SPOLL(Dsa),64) THEN        !Bit 6 set, HP35660A needs service
300     BEEP
310     OUTPUT Dsa;"STAT:USER:EVEN?"       !Read USER STATUS EVENT register
320     ENTER Dsa;User_status_reg
330        !
340        !  Check all 16 bits in the USER STATUS EVENT register
350        !  Note: Bits 10-15 can only be set via HP-IB
360     FOR Usrq_number=0 TO 15
370       IF BIT(User_status_reg,Usrq_number) THEN
380         SELECT Usrq_number
390         CASE 0
400           GOSUB Service_usrq0   !Gosub service routine for USER SRQ 0
410         CASE 1
420           GOSUB Service_usrq1   !Gosub service routine for USER SRQ 1
430         CASE 2 TO 15
440           GOSUB Service_usrqx   !Goto service routine for other USER SRQ's
450         END SELECT
460       END IF
470     NEXT Usrq_number
480     ENABLE INTR Scode            !Re-enable interrupts
490     LOCAL Dsa                    !Put the HP35660A in local mode
500   ELSE
510     BEEP
520     DISP "UNKNOWN INTERRUPT"     !Interrupt wasn't from HP35660A,
530     STOP                         !Stop the program.
540   END IF
550   RETURN
560 Service_usrq0:     !Service routine to handle USER SRQ 0
570   PRINT "User pressed SRQ 0"
580   RETURN
590 Service_usrq1:     !Service routine to handle USER SRQ 1
600   PRINT "User pressed SRQ 1"
610   RETURN
620 Service_usrqx:     !Service routine to handle other USER SRQ's
630   PRINT "USER SRQ was between 2 and 15"
640   RETURN
650   END
```

Example 10.

```
10    !BASIC program: WAI_SYNC -  Measurement synchronization
20    !
30    !This program demonstrates how to use the *WAI command to
40    !prevent execution of an HP-IB command until all previous
50    !commands have finished.  In this example, the trace display
60    !will not change to the UPPER/LOWER FORMAT until after the
70    !measurement has finished.
80    !
90    !The *WAI command does not affect program operation. The
100   !program will run to completion, sending all of the commands to
110   !to the HP35660A without waiting for them to be executed.
120   !
130   Scode=7                                   !Interface select code
140   Address=11                                !HP-IB address for HP 35660A
150   Dsa=Scode*100+Address
160   !
170   DISP "Sending HP-IB commands..."
180   OUTPUT Dsa;"SCR:FORM SING"                !Set  display format to SINGLE.
190   OUTPUT Dsa;"SWE:TIME 8"                   !Set record length to 8 seconds
200   OUTPUT Dsa;"INIT:STAT STAR"               !Start the measurement
210   OUTPUT Dsa;"*WAI"                         !Tell analyzer to wait here until
220                                             !all HP-IB commands have finished
230   OUTPUT Dsa;"SCR:FORM ULOW"                !Go to upper/lower after waiting
240   BEEP
250   DISP "Finished. Display will go to UPPER/LOWER when measurement done"
260   END
```

Example 11.

```
10    !BASIC Program:  WINDOW -- Program for user defined windows
20    !
30    !This program demonstrates how to use waveform math to implement
40    !user defined windows.  Before running the program, set the start
50    !frequency and span for the measurement. The window will be complex
60    !for zoomed measurements and real for baseband measurements.
70    !
80    Scode=7                                    !Interface select code
90    Dsa=100*Scode+11                           !HP35660A at address 11
100   DIM Window(1023)                           !1024 pt array for window
110   !
120   ASSIGN @Dsa_off TO Dsa;FORMAT OFF          !IO path for binary data
130   !
140   OUTPUT Dsa;"USER:EXPR F1,(TIME1/TIME1)"    !Define a unitless time trace
150   OUTPUT Dsa;"TRAC:A:RES F1"                 !Set trace A to Math F1
160   OUTPUT Dsa;"TRAC:HEAD:XOR?"
170   ENTER Dsa;Xor                              !Read the start time
180   OUTPUT Dsa;"TRAC:HEAD:XINC?"
190   ENTER Dsa;Xinc                             !Read the time interval
200   OUTPUT Dsa;"TRAC:HEAD:YPO?"                !Read Y points/bin
210   ENTER Dsa;Ypo                              !Data is complex if Ypo=2
220   !
230   Gen_window(Window(*),Ypo)                  !Build the window function
240   !
250   DISP "Writing window to trace A..."
260   OUTPUT Dsa;"TRAC:HEAD:XOR ";Xor            !Define the start time
270   OUTPUT Dsa;"TRAC:HEAD:XINC ";Xinc          !Define the time interval
280   OUTPUT Dsa;"TRAC:HEAD:YPO ";Ypo            !1 or 2  Y points/point
290   OUTPUT Dsa;"TRAC:DATA";                    !Put window data into trace A
300   OUTPUT Dsa USING "#,A,D,4D";"#",4,1024*8   !Send a header '#46416'
310   OUTPUT @Dsa_off;Window(*),CHR$(10)         !Send data and block terminator
320   !
330   OUTPUT Dsa;"DISP:A:AXIS REAL"              !Display the real part
340   OUTPUT Dsa;"DISP:A:SCAL:AUTO:SING"         !Autoscale the display
350   DISP "Saving the window in 'RAM:HANNING'..."
360   OUTPUT Dsa;"MMEM:STOR:TRAC:A 'RAM:HANNING';*OPC?"  !Save the window
370   ENTER Dsa;Opc                              !Wait for save to finish
380   !
390   !Use the window to do a spectrum measurement.  Keep TIME1 first in
400   !the math expression for correct x-axis start/stop values.
410   OUTPUT Dsa;"USER:EXPR F1,(FFT(TIME1*'RAM:HANNING'))"  !Define math F1
420   OUTPUT Dsa;"TRAC:A:RES F1"                 !Set trace A to Math F1
430   DISP "Starting the measurement..."
440   OUTPUT Dsa;"INIT:STAT STAR"                !Start the measurement
450   OUTPUT Dsa;"DISP:A:AXIS LOGM;*WAI;:DISP:A:SCAL:AUTO:SING"
460   !
470   LOCAL Dsa                                  !Return the HP35660A to LOCAL
480   DISP "Program finished"
490   END
500   !
510   SUB Gen_window(Wind(*),Ypo)   !Subroutine to generate HANNING window
520     RAD                                      !Work in radians
530     DISP "Generating data for HANNING window..."
540     Const=2*PI/1023                          !Do this calculation once
550     IF Ypo=1 THEN                            !Window is real.  1024 x 1
560       FOR I=0 TO 1023
570         Wind(I)=1.0+COS(I*Const+PI)          !Generate window function
580       NEXT I
590     ELSE                                     !Window is complex. 512 x 2
600       FOR I=0 TO 1023 STEP 2
610         Wind(I)=1.0+COS(I*Const+PI)          !Generate real point
620         Wind(I+1)=0.0                        !Set imaginary point to 0
630       NEXT I
640     END IF
650   SUBEND
```

Chapter 7
Command Reference

Introduction

This command reference describes all of the HP 35660A's HP-IB commands. Figure 7-1 shows you the fields included in the command descriptions:

1. A summary of important command attributes.
2. One or more example statements that incorporate the command.
3. Command and/or query syntax.
4. The format of returned data (for commands that have query forms).
5. A detailed command description.

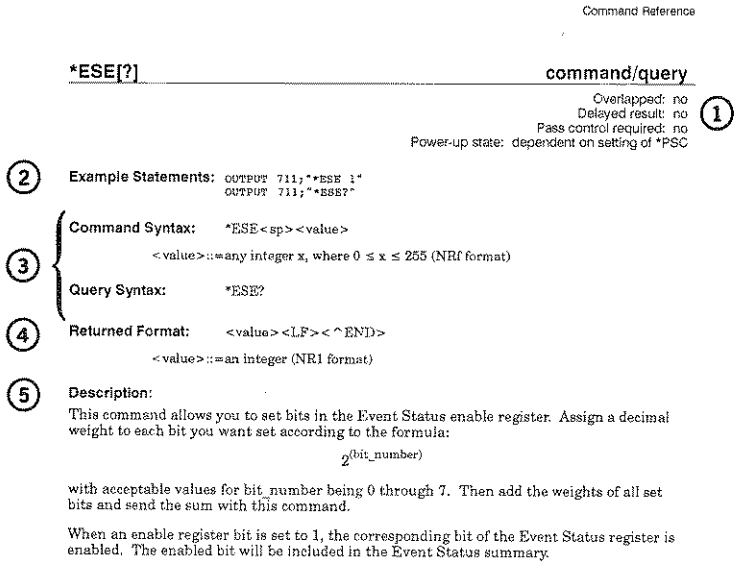


Figure 7-1. Sample Command Description

The overlapped, delayed result, and pass control attributes are described in Chapter 2, "Behavior in an HP-IB System." All example statements are written in HP BASIC 5.0 for an HP Series 200 computer. Returned Format describes the format of returned data when SYST:HEAD is OFF. See the SYST:HEAD command for information about the format of returned data when SYST:HEAD is ON.

Conventions

Syntax and returned format descriptions use the following conventions:

- < > Angle brackets enclose the names of syntactic items that need further definition. The definition will be included in accompanying text or in the summary of common definitions that follows this section.
- ::= "is defined as" When two items are separated by this symbol, the second item can replace the first in any statement that contains the first item. For example, A::=B indicates that B can replace A in any statement that contains A.
- | "or" When items in a list are separated by this symbol, one and only one of the items can be chosen from the list. For example, A|B indicates that A or B can be chosen, but not both.
- ... An ellipsis (trailing dots) is used to indicate that the preceding element may be repeated one or more times.
- [ ] Square brackets indicate that the enclosed items are optional.
- ~ ~ Tildes surround items that are understood as the default when none of the items in a list are selected.
- { } Braces are used to group items into a single syntactic element. They are most often used to enclose lists and to enclose elements that are followed by an ellipsis.

In addition, the case of letters in the command mnemonics is significant. Mnemonics that are longer than four characters can have a short form or a long form. The analyzer accepts either form. Upper-case letters show the short form of a command mnemonic. For more information, see "Command Abbreviation" in Chapter 3.

Common Definitions

Syntax and returned format descriptions have the following definitions in common:

- <LF> is the line feed character (ASCII decimal 10).
- <^END> is assertion of the HP-IB END message while the last byte of data is on the bus.
- <sp> is the space character (ASCII decimal 32).

Common Commands

This section describes all of the IEEE 488.2 common commands that are implemented in the HP 35660A. An important property of all common commands is that you can send them without regard to a program message's position in the command tree. For more information on the analyzer's command tree, see Chapter 3, "Programming with Hierarchical Commands."

*CAL? query

Overlapped: no
Delayed result: no
Pass control required: no
Power-up state: 0

Example Statement: `OUTPUT 711;"*CAL?"`

Query Syntax: *CAL?

Returned Format: {0|1}<LF><^END>

Description:

This query causes the analyzer to recalibrate. If the calibration is completed without error, the analyzer returns 0. If the calibration is not completed without error, the analyzer returns 1.

*CLS command

Overlapped: no
Delayed result: no
Pass control required: no
Power-up state: not applicable

Example Statement: `OUTPUT 711;"*CLS"`

Command Syntax: *CLS

Description:

This command clears the Status Byte register. It does so by clearing (resetting to 0) all bits in the following event registers:

- Data Integrity event register
- Device Status event register
- User Status event register
- Event Status register

In addition, *CLS clears the error queue and cancels any preceding *OPC command or query. This ensures that bit 0 of the Event Status register will not be set and that no response will be placed in the analyzer's output queue when pending overlapped commands are completed.

*CLS does not change the current state of enable registers or transition registers.

NOTE *CLS should be sent immediately following a Program Message Terminator. This guarantees that the Status Byte's Message Available (MAV) and Master Summary Status bits will be cleared.

See Chapter 5 for more information on the Status Byte register.

*ESE[?] command/query

Overlapped: no
Delayed result: no
Pass control required: no
Power-up state: dependent on setting of *PSC

Example Statements:
```
OUTPUT 711;"*ESE 1"
OUTPUT 711;"*ESE?"
```

Command Syntax: *ESE<sp><value>

<value>::=any integer x, where 0 ≤ x ≤ 255 (NRf format)

Query Syntax: *ESE?

Returned Format: <value><LF><^END>

<value>::=an integer (NR1 format)

Description:

This command allows you to set bits in the Event Status enable register. Assign a decimal weight to each bit you want set according to the formula:

$$2^{(bit\_number)}$$

with acceptable values for bit_number being 0 through 7. Then add the weights of all set bits and send the sum with this command.

When an enable register bit is set to 1, the corresponding bit of the Event Status register is enabled. The enabled bit will be included in the Event Status summary.

The Event Status summary is reported to bit 5 of the Status Byte register. Bit 5 is only set if both of the following are true:

- One or more bits in the Event Status register are set
- At least one of the set bits is enabled by a corresponding bit in the Event Status enable register

The option last specified with *ESE is saved in nonvolatile memory when you send the SYST:SAVE command. It can be recalled at power-up, depending on the setting of *PSC. When the setting of *PSC is 0 at power-up, all bits in the Event Status enable register are set according to the saved *ESE value. When the setting of *PSC is 1 at power-up, all bits in the Event Status enable register are initialized to 0. The current setting of bits is not modified when you send the *RST command.

The query returns the current state of the Event Status enable register. The state is returned as a sum of the decimal weights of all set bits.

For more information on the Event Status register set, see Chapter 5.

*ESR? query

Overlapped: no
Delayed result: no
Pass control required: no
Power-up state: 128

Example Statement: `OUTPUT 711;"*ESR?"`

Query Syntax: *ESR?

Returned Format: <value><LF><^END>

<value>::=any integer x, where $0 \leq x \leq 255$ (NR1 format)

Description:

This query returns the current state of the Event Status register. The state is returned as a sum of the decimal weights of all set bits. The decimal weight for each bit is assigned according to the formula:

$$2^{(bit\_number)}$$

with acceptable values for bit_number being 0 through 7.

The register is cleared after being read by this query.

A bit in the Event Status register is set to 1 when the condition that bit monitors becomes true. A set bit remains set, regardless of further changes in the condition it monitors, until the Event Status register is:

- Read by this query or
- Cleared by the *CLS command.

For more information on the Event Status register set, see Chapter 5.

*IDN? query

Overlapped: no
Delayed result: no
Pass control required: no
Power-up state: instrument dependent

Example Statement: `OUTPUT 711;"*IDN?"`

Query Syntax: *IDN?

Returned Format: HEWLETT-PACKARD,35660A,<serial_num>,<revision_num><LF><^END>

<serial_num>::=10 ASCII characters

<revision_num>::=7 ASCII characters

Description:

The response to this query uniquely identifies your analyzer.

*OPC[?] command/query

Overlapped: no
Delayed result: no
Pass control required: no
Power-up state: not applicable

Example Statements:
```
OUTPUT 711;"*OPC"
OUTPUT 711;"*OPC?"
```

Command Syntax: *OPC

Query Syntax: *OPC?

Returned Format: 1<LF><^END>

Description:

Use *OPC or *OPC? if you want to know when all pending overlapped commands have been completed.

Most commands that you send to the analyzer are processed sequentially. A sequential command will hold off the processing of any subsequent commands until it has been completely processed. However, some commands will not hold off the processing of subsequent commands; they are referred to as overlapped commands.

The analyzer uses an operation complete (OPC) flag to keep track of overlapped commands that are still pending (that is, not completed). The OPC flag is reset to 0 when an overlapped command is pending. It is set to 1 when no overlapped commands are pending. You can not read the OPC flag directly, but you can use *OPC and *OPC? to tell when the flag is set to 1.

If you use *OPC, bit 0 of the Event Status register is set to 1 when the OPC flag is set to 1. This allows the analyzer to generate a service request when all pending overlapped commands are completed (assuming you have enabled bit 0 of the Event Status register and bit 5 of the Status Byte register).

If you use *OPC?, 1 is placed in the output queue when the OPC flag is set to 1. This allows you to effectively pause the controller until all pending overlapped commands are completed. It must wait until the response is placed in the queue before it can continue.

NOTE The *CLS and *RST commands cancel any preceding *OPC command or query. Pending overlapped commands are still completed, but you can no longer determine when. Two HP-IB bus management commands Device Clear (DCL) and selected Device Class (SDC) will also cancel any preceding *OPC command or query.

---

*OPT? query

Overlapped: no
Delayed Result: no
Pass Control Required: no
Power-up state: instrument dependent

Example statement:

```
OUTPUT 711; "*OPT?"
```

Query Syntax: *OPT?

Return Format: {0|<option>[<option>]}<LF><^END>

<option>::= a series of ASCII characters describing an instrument option (never more than 255 characters).

Description:

This query allows the analyzer to report any options it contains. For example, if your analyzer contains an internal disc drive, it returns DISC in response to this query.

The analyzer returns 0 if it contains no special options.

*PCB command

Overlapped: no
Delayed result: no
Pass control required: no
Power-up state: saved in nonvolatile memory

Example Statement: `OUTPUT 711;"*PCB 21"`

Command Syntax: *PCB<sp><value>[,<value>]

value::=an integer (NRf format)

Description:

Use this command to specify the address of a controller that is temporarily passing control of the HP-IB to the analyzer. When the analyzer completes the operation that required it to have control of the bus, it automatically passes control back to the controller at this address.

The optional second <value> is only used for controllers that have extended addressing. It is interpreted as the secondary address of the controller.

The option last specified with this command is saved in nonvolatile memory when you send the SYST:SAVE command. This means that when you turn the analyzer off and then back on, the specified controller address does not change.

*PSC[?] command/query

Overlapped: no
Delayed result: no
Pass control required: no
Power-up state: saved in nonvolatile memory

Example Statements:
```
OUTPUT 711;"*PSC 1"
OUTPUT 711;"*PSC?"
```

Command Syntax: *PSC<sp>{0|1}

Query Syntax: *PSC?

Returned Format: {0|1}<LF><^END>

Description:

This command allows you to specify whether or not the Service Request enable register and the Event Status enable register should be cleared (all bits reset to 0) at power-up.

The settings of the Service Request enable register and the Event Status enable register are saved in nonvolatile memory when the analyzer is turned off. These settings can be recalled when you turn the analyzer on, but only if the Power-on Status Clear (PSC) flag is reset to 0. When the PSC flag is set to 1, the two enable registers are cleared at power-up. Use *PSC to specify the setting of the PSC flag.

The option last specified with *PSC is saved in nonvolatile memory when you send the SYST:SAVE command. This means that when you turn the analyzer off and then back on, the state of *PSC does not change.

If you want the analyzer to generate a service request at power-up, bit 7 of the Event Status register and bit 5 of the Status Byte register must be enabled. This is only possible if the PSC flag is set to 0.

The query returns the current setting of the PSC flag.

*RST command

Overlapped: yes
Delayed result: no
Pass control required: no
Power-up state: not applicable

Example Statement: `OUTPUT 711;"*RST"`

Command Syntax: *RST

Description:

This command returns the analyzer to its power-up state. In addition, it cancels any *OPC or *OPC? and clears all event registers.

The following are not affected by this command:

- All states saved in nonvolatile memory
- The state of all enable and transition registers
- The state of INP:UNIT:EU:MULT
- The state of INP:UNIT:EU:NAME
- The state of CAL:AUTO
- The state of SYST:BEEP
- Calibration constants
- Math functions and constants
- The input buffer and output queue
- Limit and data table entries

For a list of the states saved in nonvolatile memory, see the SYST:SAVE command.

*SRE[?] command/query

Overlapped: no
Delayed result: no
Pass control required: no
Power-up state: dependent on setting of *PSC

Example Statement:
```
OUTPUT 711;"*SRE 160"
OUTPUT 711;"*SRE?"
```

Command Syntax: *SRE<sp><value>

<value>::=any integer x, where $0 \leq x \leq 255$ (NRf format)

Query Syntax: *SRE?

Returned Format: <value><LF><^END>

<value>::=an integer (NR1 format)

Description:

This command allows you to set bits in the Service Request enable register. Assign a decimal weight to each bit you want set according to the formula:

$$2^{(bit\_number)}$$

with acceptable values for bit_number being 0 through 7. Then add the weights of all set bits and send the sum with this command.

NOTE The analyzer ignores the setting you specify for bit 6 of the Service Request enable register. This is because the corresponding bit of the Status Byte register is always enabled.

The analyzer requests service from the active controller when one of the following occurs:

- A bit in the Status Byte register changes from 0 to 1 while the corresponding bit of the Service Request enable register is set to 1.
- A bit in the Service Request enable register changes from 0 to 1 while the corresponding bit of the Status Byte register is set to 1.

The option last specified with *SRE is saved in nonvolatile memory when you send the SYST:SAVE command. It can be recalled at power-up, depending on the setting of *PSC. When the setting of *PSC is 0 at power-up, all bits in the Service Request enable register are set according to the saved *SRE value. When the setting of *PSC is 1 at power-up, all bits in the Service Request enable register are initialized to 0. The current setting of bits is not modified when you send the *RST command.

The query returns the current state of the Service Request enable register. The state is returned as a sum of the decimal weights of all set bits.

*STB? query

Overlapped: no
Delayed result: no
Pass control required: no
Power-up state: variable

Example Statement: `OUTPUT 711;"*STB?"`

Query Syntax: `*STB?`

Returned Format: <value><LF><^END>

<value>::=any integer x, where $0 \leq x \leq 255$ (NR1 format)

Description:

This query returns the current state of the Status Byte register. The state is returned as a sum of the decimal weights of all set bits. The decimal weight for each bit is assigned according to the formula:

$$2^{(bit\_number)}$$

with acceptable values for bit_number being 0 through 7.

The setting of bits is not affected by this query. To reset the bits in the Status Byte register, you must use the *CLS command.

Bits in the Status Byte register are defined as follows:

- Bit 0 summarizes all enabled bits of the User Status event register.
- Bits 1, 2, and 3 are reserved.
- Bit 4 is the Message Available (MAV) bit. It is set whenever there is something in the analyzer's output queue.
- Bit 5 summarizes all enabled bits of the Event Status register.
- Bit 6, when read with this query (*STB?), acts as the Master Summary Status (MSS) bit. It summarizes all enabled bits of the Status Byte register. (Bit 6 acts as the Request Service (RQS) bit when it is read by a serial poll.)
- Bit 7 summarizes all enabled bits of the Device Status event register.

For more information on the Status Byte register, see Chapter 5.

*TRG command

Overlapped: no
Delayed result: no
Pass control required: no
Power-up state: not applicable

Example Statement: `OUTPUT 711;"*TRG"`

Command Syntax: *TRG

Description:

This command triggers the analyzer if the following two things are true:

- The trigger source must be the HP-IB (TRIG:SOUR BUS)
- The analyzer must be ready to trigger. (Bit 2 of the Device Status condition register must be set.)

The *TRG command has the same effect as TRIG:IMM. It also has the same effect as the HP-IB bus management command Group Execute Trigger (GET).

*TST? query

Overlapped: no
Delayed result: no
Pass control required: no
Power-up state: 0

Example Statement: `OUTPUT 711;"*TST?"`

Query Syntax: *TST?

Returned Format: {0|1}<LF><^END>

Description:

This query invokes a self-test that verifies proper operation of the analyzer's hardware.

When you send the query, the analyzer self-calibrates and then compares the calibration results to specified limits. If the results are within the specified limits, the analyzer returns 0. If the results exceed the specified limits, the analyzer returns 1.

*WAI command

Overlapped: no
Delayed result: no
Pass control required: no
Power-up state: not applicable

Example Statement: `OUTPUT 711;"*WAI"`

Command Syntax: *WAI

Description:

Use *WAI to hold off the processing of subsequent commands until all pending overlapped commands have been completed.

Most commands that you send to the analyzer are processed sequentially. A sequential command will hold off the processing of any subsequent commands until it has been completely processed. However, some commands will not hold off the processing of subsequent commands; they are referred to as overlapped commands. *WAI ensures that overlapped commands will be completely processed before subsequent commands (those sent after *WAI) are processed.

Device-Specific Commands

ARM subsystem

Description:

This subsystem contains commands and queries related to the analyzer's trigger arming functions. See the TRIG subsystem for commands related to other triggering functions.

ARM[:IMMediate] command

Overlapped: no
Delayed result: no
Pass control required: no
Power-up state: not applicable

Example Statements:

```
OUTPUT 711;"ARM"
OUTPUT 711;"Arm:Immediate"
```

Command Syntax: ARM[:IMMediate]

Description:

This command enables triggering when the following two things are true:

- Manual arming is selected.
- Bit 3 of the Device Status condition register (ready-for-arm) is set to 1.

After sending the command, triggering occurs when the appropriate trigger signal is received. The command is ignored when automatic arming is selected.

See ARM:SOUR for more information on arming modes and TRIG:SOUR for information on selecting the trigger signal.

ARM:SOURce[?] command/query

Overlapped: no
Delayed result: no
Pass control required: no
Power-up state: FREE

Example Statements:

```
OUTPUT 711;"ARM:SOUR FREE"
OUTPUT 711;"ARM:SOURCE HOLD"
OUTPUT 711;"Arm:Sour?"
```

Command Syntax: ARM:SOURce<sp>{FREErun|HOLD}

Query Syntax: ARM:SOURce?

Returned Format: {FREE|HOLD}<LF><^END>

Description:

This command allows you to select one of two modes for arming the trigger. The modes are:

- Automatic arming (ARM:SOUR FREE)
- Manual arming (ARM:SOUR HOLD)

In order for the analyzer to make a measurement, its trigger must be armed before a trigger signal is received. For non-averaged measurements, the trigger must be armed before each measurement. For averaged measurements, the trigger must be armed before each new time record.

When you start a measurement with automatic arming selected, the analyzer waits for the digital filters to settle and then triggers as soon as a trigger signal is received. When the measurement or average is completed, the trigger is automatically re-armed.

When you start a measurement with manual arming selected, the analyzer waits for the digital filters to settle and then waits for you to send the ARM:IMM command. Once both of these conditions are met, the analyzer triggers as soon as a trigger signal is received. When the measurement or average is completed, you must once again send the ARM:IMM command to re-arm the trigger.

The query returns an ASCII string that indicates whether FREErun or HOLD is selected.

See TRIG:SOUR for information on selecting the trigger signal.

AVERage subsystem

Description:

This subsystem contains commands related to the analyzer's data averaging functions.

AVER:COUNt[?] command/query

Overlapped: no
Delayed result: yes
Pass control required: no
Power-up state: 10

Example Statements:

```
OUTPUT 711;"AVER:COUN 5"
OUTPUT 711;"Average:Count 100"
OUTPUT 711;"AVER:COUN?"
```

Command Syntax: AVERage:COUNt<sp><value>

<value>::=a single integer from 1 to 99999 (NRf format)

Query Syntax: AVERage:COUNt?

Returned Format: <value><LF><^END>

<value>::=a single integer (NR1 format)

Description:

The value sent with this command is used in different ways, depending on the kind of average weighting you have specified. When stable weighting is specified (AVER:WEIG STAB), the value you send with this command determines the number of averages required to complete a measurement. When exponential weighting is specified (AVER:WEIG EXP), the value you send with this command determines two things:

- The number of averages required to complete the first phase of an exponentially averaged measurement
- The weighting factors for new and old data during the second phase of an exponentially averaged measurement

If peak averaging is selected (AVER:TYPE PEAK), or if averaging is turned off (AVER:STAT OFF), the value sent with this command does not affect the measurement.

The value sent with this command affects the setting of the measuring bit in the Device Status condition register. For more information, see Chapter 5, "Using the HP 35660A's Status Registers."

The query returns a value indicating the current number of averages selected.

AVER:DISPlay — selector

Description:

This command only selects the AVER:DISP subsystem. Sending AVER:DISP alone does nothing.

AVER:DISP:RATE[?] — command/query

Overlapped: no
Delayed result: yes
Pass control required: no
Power-up state: 5

Example Statements:

```
OUTPUT 711;"aver:disp:rate 10"
OUTPUT 711;"AVERAGE:DISPLAY:RATE 5"
OUTPUT 711;"AVER:DISP:RATE?"
```

Command Syntax: AVERage:DISPlay:RATE<sp><value>

<value>::=a single integer from 1 to 99999 (NRf format)

Query Syntax: AVERage:DISPlay:RATE?

Returned Format: <value><LF><^END>

<value>::=a single integer (NR1 format)

Description:

When fast averaging is turned on (AVER:DISP:RATE:STAT ON) you can specify an interval for display updates. This is the command you use to specify that interval. For example, if you send AVER:DISP:RATE 5, the display is updated once every 5 averages.

The value specified with this command is not used if fast averaging is turned off (AVER:DISP:RATE:STAT OFF).

The query returns a value that indicates the current display update rate.

AVER:DISP:RATE:STATe[?] command/query

Overlapped: no
Delayed result: yes
Pass control required: no
Power-up state: 0

Example Statements:
```
OUTPUT 711;"AVER:DISP:RATE:STAT 1"
OUTPUT 711;"Average:Display:Rate:State Off"
OUTPUT 711;"aver:disp:rate:stat?"
```

Command Syntax: AVERage:DISPlay:RATE:STATe<sp>{OFF|ON|0|1}

Query Syntax: AVERage:DISPlay:RATE:STATe?

Returned Format: {0|1}<LF><^END>

Description:

Use this command to turn fast averaging off and on. When fast averaging is off, the display is updated each time one average is taken.

When fast averaging is on, the display is updated each time a specified number of averages is taken. The number can be from 1 to 99,999 and is specified with the AVER:DISP:RATE command. For example, if fast averaging is on (AVER:DISP:RATE:STAT ON), and the value of AVER:DISP:RATE is 5, the display is updated every 5 averages. If the number specified in AVER:DISP:RATE is larger than the number of averages required to complete your measurement, the display is only updated when the measurement is paused or completed.

The query returns 0 if fast averaging is off, 1 if it is on.

AVER:INITialize command

Overlapped: no
Delayed result: yes
Pass control required: no
Power-up state: not applicable

Example Statements:
```
OUTPUT 711;"Aver:Init"
OUTPUT 711;"AVERAGE:INITIALIZE"
```

Command Syntax: AVERage:INITialize

Description:

This command sets a flag so that the next INIT:STAT RUN command will start a new running average. For example, if a measurement is paused and you send AVER:INIT followed by INIT:STAT RUN, a new running average is started after the old one is discarded. However, if a measurement is paused and you only send INIT:STAT RUN, the paused measurement continues from where it was stopped and new data is averaged in with the old running average.

AVER:OVERlap[?] command/query

Overlapped: no
Delayed result: yes
Pass control required: no
Power-up state: 0

Example Statements:

```
OUTPUT 711;"AVER:OVER .1"
OUTPUT 711;"average:overlap 90PCT"
OUTPUT 711;"AVERAGE:OVERLAP?"
```

Command Syntax: AVERage:OVERlap<sp>{{<percent>PCT}|<fraction>}

<percent>::=an integer from 0 to 99 (NRf format)

<fraction>::=a decimal number from .00 to .99 in increments of .01 (NRf format)

Query Syntax: AVERage:OVERlap?

Returned Format: <fraction><LF><^END>

<fraction>::=a decimal number (NR2 format)

Description:

Under certain conditions, data points from the end of one time record can be reused at the beginning of the next time record. This results in the overlapping of time records. Use the AVER:OVER command to specify the maximum amount of time record overlap you want to allow.

Overlapping becomes possible when the instrument takes more time to collect time records than it does to process them. This occurs at narrower frequency spans. At spans narrow enough to allow the requested amount of overlapping, time records will be overlapped.

You can specify overlap either as a percentage or as a fraction of the time record length. AVER:OVER 22PCT is the same as AVER:OVER 0.22. In either case, the value you send is rounded to the nearest allowable percentage (an integer between 0 an 99). You can step the current overlap setting up or down 1% by sending AVER:OVER UP or AVER:OVER DOWN.

The query returns a value that indicates the amount of overlap currently specified. The value is returned in fractional form.

AVER[:STATe][?] command/query

Overlapped: no
Delayed result: yes
Pass control required: no
Power-up state: 0

Example Statements:

```
OUTPUT 711;"AVER OFF"
OUTPUT 711;"AVERAGE:STATE 1"
OUTPUT 711;"Aver?"
```

Command Syntax: AVERage[:STATe]<sp>{OFF|ON|0|1}

Query Syntax: AVERage[:STATe]?

Returned Format: {0|1}<LF><^END>

Description:

Use this command to turn averaging off and on.

The query returns 0 if averaging is off, 1 if averaging is on.

See the following for more information:

- AVER:TYPE – for selecting an averaging type.
- AVER:WEIG – for specifying how averaged data should be weighted.
- AVER:COUN – for specifying the number of averages

AVER:TYPE[?] command/query

Overlapped: no
Delayed result: yes
Pass control required: no
Power-up state: RMS

Example Statements:
```
OUTPUT 711;"AVER:TYPE RMS"
OUTPUT 711;"Average:Type Peak"
OUTPUT 711;"Aver:Type?"
```

Command Syntax: AVERage:TYPE<sp>{PEAK|RMS|VECTor}

Query Syntax: AVERage:TYPE?

Returned Format: {PEAK|RMS|VECT}<LF><^END>

Description:

This is one of two commands that affect the way running averages of measurement data are calculated. The other command is AVER:WEIG.

You can specify one of three options with this command:

- Rms – root mean square averaging of the last n power spectra or linear averaging of the last n cross spectra
- Vector – vector averaging of the last n linear spectra
- Peak hold – point by point maximum of the last n power spectra, not available for cross spectra

With rms averaging selected (AVER:TYPE RMS), you get a good approximation all input signal components, including noise. Each frequency bin is averaged separately.

With vector averaging (AVER:TYPE VECT) and an appropriate trigger source selected, noise components tend to cancel. This allows you to resolve smaller periodic signals. Each frequency bin is averaged separately.

With peak hold selected (AVER:TYPE PEAK), the peak value for each frequency bin is retained each time a new power spectrum is acquired. The value of AVER:COUN is not used to stop the acquisition of new spectra, so they are acquired continuously until the measurement is paused.

When stable weighting is selected, rms and vector averaging stop when the specified number of averages is acquired. When exponential weighting is selected, rms and vector averaging continue indefinitely until the measurement is paused.

The query response indicates which type of averaging is currently selected.

AVER:WEIGhting[?] command/query

Overlapped: no
Delayed result: yes
Pass control required: no
Power-up state: STAB

Example Statements:
```
OUTPUT 711;"aver:weig stab"
OUTPUT 711;"Average:Weighting Exponential"
OUTPUT 711;"AVER:WEIG?"
```

Command Syntax: AVERage:WEIGhting<sp>{EXPonential|STABle}

Query Syntax: AVERage:WEIGhting?

Returned Format: {EXP|STAB}<LF><^END>

Description:

This command allows you to specify how averaged data will be weighted. The options are:

- Stable (or uniform) weighting (AVER:WEIG STAB)
- Exponential weighting (AVER:WEIG EXP)

With stable averaging selected, each spectrum included in the running average is weighted equally. Also, the measurement stops when the specified number of averages has been acquired.

With exponential averaging selected, there are two distinct phases to the averaging process. During the first phase, each spectrum included in the running average is weighted equally, as in stable averaging. During the second phase, new and old data are weighted as follows:

$$[(1/N)\times new]+[((N-1)/N)\times old]$$

Where: N is the value of AVER:COUN (number of averages)
new is the most recently acquired spectrum
old is the data in the running average

The first phase of an exponentially averaged measurement continues until the running average includes the number of spectra specified in AVER:COUN (number of averages). The second phase continues indefinitely until the measurement is paused.

The setting of AVER:WEIG is not used if peak hold averaging (AVER:TYPE PEAK) is selected.

The query returns a mnemonic that indicates the type of weighting selected: EXP for exponential or STAB for stable weighting.

CALibration subsystem

Description:

This subsystem contains commands related to calibration of the analyzer.

CAL[:ALL]? query

Overlapped: no
Delayed result: no
Pass control required: no
Power-up state: not applicable

Example Statement: `OUTPUT 711;"CAL?"`

Query Syntax: CALibration[:ALL]?

Returned Format: <value><LF><^END>

<value>::=an integer (NR1 format)

Description:

The instrument performs a full calibration when you send this query. The query response is a 0 if the calibration is successful. The response is a non-zero integer if the calibration fails, with the integer being an error number.

The calibration routine performed when you send this query is the same as the calibration routine performed when you send the CAL:SING command.

CAL:AUTO[?] command/query

Overlapped: no
Delayed result: yes
Pass control required: no
Power-up state: 1

Example Statements:

```
OUTPUT 711;"CAL:AUTO OFF"
OUTPUT 711;"calibration:auto 1"
OUTPUT 711;"Cal:Auto?"
```

Command Syntax: CALibration:AUTO<sp>{OFF|ON|0|1}

Query Syntax: CALibration:AUTO?

Returned Format: {0|1}<LF><^END>

Description:

Use this command to enable and disable the analyzer's autocalibration routine. The routine causes the analyzer to calibrate automatically at power-up, several times during the first hour of operation, and once each hour after that.

NOTE The autocalibration routine does not interrupt an averaged measurement in progress.

When you turn autocalibration off (CAL:AUTO OFF), the analyzer is only recalibrated when you send the CAL:SING command or the CAL:ALL query. Calibration always occurs automatically at power-up.

The query returns 0 if autocalibration is disabled, 1 if it is enabled.

CAL:CLEar command

Overlapped: no
Delayed result: yes
Pass control required: no
Power-up state: not applicable

Example Statements:
```
OUTPUT 711;"Cal:Cle"
OUTPUT 711;"CALIBRATION:CLEAR"
```

Command Syntax: CALibration:CLEar

Description:

This command clears all calibration constants until the next calibration occurs. While the calibration constants are cleared, data from the current measurement is uncalibrated.

If autocalibration is enabled (CAL:AUTO ON), the calibration constants can remain cleared for as long as one hour. If autocalibration is disabled, the constants remain cleared until you send CAL:ALL? or CAL:SING.

Bit 2 of the Data Integrity Condition register indicates whether or not the calibration constants are currently cleared.

CAL:SINGle command

Overlapped: yes
Delayed result: no
Pass control required: no
Power-up state: not applicable

Example Statements:
```
OUTPUT 711;"CAL:SING"
OUTPUT 711;"Calibration:Single"
```

Command Syntax: CALibration:SINGle

Description:

This command causes the analyzer to recalibrate immediately. The calibration occurs whether the autocalibration routine is enabled or disabled (CAL:AUTO ON or OFF). The analyzer's measurement activities are suspended during the calibration.

CAL:TRACe[?] command/query

Overlapped: no
Delayed result: yes
Pass control required: no
Power-up state: 0

Example Statements:

```
OUTPUT 711;"CAL:TRAC 1"
OUTPUT 711;"Calibration:trace OFF"
OUTPUT 711;"CAL:TRAC?"
```

Command Syntax: CALibration:TRACe<sp>{OFF|ON|0|1}

Query Syntax: CALibration:TRACe?

Returned Format: {0|1}<LF><^END>

Description:

This command allows you to display the calibration constants that will be used for a particular measurement setup. To display the constant for a setup, you must do all of the following:

1. Specify the measurement setup
2. send CAL:TRAC ON
3. send INIT:STAT STAR

The query returns 0 if the calibration constants are not being displayed, 1 if they are.

CONFigure subsystem

Description:

The single command in this subsystem is used to switch the analyzer between its one-channel and two-channel operating modes.

CONF:TYPE[?] command/query

Overlapped: no
Delayed result: yes
Pass control required: no
Power-up state: SPEC

Example Statements:
```
OUTPUT 711;"conf:type spec"
OUTPUT 711;"CONFIGURE:TYPE NETWORK"
OUTPUT 711;"Conf:Type?"
```

Command Syntax: CONFigure:TYPE<sp>{NETWork|SPECtrum}

Query Syntax: CONFigure:TYPE?

Returned Format: {NETW|SPEC}<LF><^END>

Description:

Use this command to select the analyzer's one-channel or two-channel operating mode.

The one-channel mode is selected with CONF:TYPE SPEC. In this mode, channel 1 can analyze signal components up to 102.4 kHz. Channel 2 is not used at all. The following data can be displayed in this mode:

- Channel 1 time (TRAC:RES TIME1)
- Channel 1 spectrum, linear or power (TRAC:RES SPEC1)
- Channel 1 power spectral density (TRAC:RES PSD1)
- Functions 1-5 (TRAC:RES F1-5)
- Constants 1-5 (TRAC:RES K1-5)
- Recalled traces (MMEM:LOAD:TRAC <file_spec>)

The two-channel mode is selected with CONF:TYPE NETW. In this mode, channels 1 and 2 can both analyze signal components up to 51.2 kHz. When this mode is selected, all data available in the one-channel mode and the following additional data can be displayed:

- Channel 2 time (TRAC:RES TIME2)
- Channel 2 spectrum, linear or power (TRAC:RES SPEC2)
- Channel 2 power spectral density (TRAC:RES PSD2)
- Frequency response (TRAC:RES FRES)
- Coherence (TRAC:RES COH)
- Cross spectrum (TRAC:RES CSP)

The query returns SPEC if the one-channel mode is selected, NETW if the two-channel mode is selected.

DISPlay subsystem

Description:

This subsystem has three main purposes:

- It provides commands for setting x-axis and y-axis scaling on the two displays.
- It provides access to the limit table data and many of the limit table functions.
- It provides access to the displayed data (data that has already been transformed into the current display coordinates). See the TRAC subsystem for access to the raw data from which the displayed data is derived.

The following diagram shows you the difference between data available in the TRAC subsystem and the DISP subsystem:

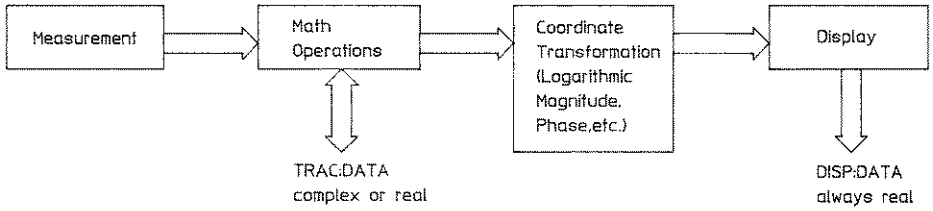


Figure 7-2. Flow of Measurement Data

After measurement data is collected, any specified math operations are performed. Data is then transformed into the specified coordinate system and sent to the display. TRAC:DATA provides access to the raw measurement data after math operations have been performed. This data can be either complex or real. DISP:DATA provides access to the displayed data, after the coordinate transformation. This data is always real.

NOTE Both TRAC:DATA and DISP:DATA allow you to take measurement data out of the analyzer. However, only TRAC:DATA allows you to put measurement data back into the analyzer.

With a few exceptions, display commands must be directed to one of the two displays: A or B. To specify a display, insert one of the following items between DISPLAY or DISP and the rest of the command:

- :A – as in DISP:A:GRAT ON
- :B – as in DISPLAY:B:LIM:BEEP?
- 1 – as in DISPLAY1:X:SPACING LIN
- 2 – as in DISP2:Y:SCAL:STAR?

Using :A or 1 directs the command to display A. Using :B or 2 directs the command to display B. If you don't explicitly specify one of the displays, the command is directed to display A.

NOTE The display to which you direct a command becomes the active display.

When HP Instrument BASIC is installed in the analyzer, additional commands are added to this subsystem. For information on these commands, see Appendix D in the HP Instrument BASIC Programming Reference.

DISP:DATA? query

Overlapped: no
Delayed result: no
Pass control required: no
Power-up state: not applicable

Example Statement: `OUTPUT 711;"DISP:DATA?"`

Query Syntax: DISPlay[<spec>]:DATA?

<spec>::=~:A~|:B|1|2

Returned Format: <block_data>

<block_data> takes one of two forms, depending on whether the data is ASCII-encoded or binary-encoded. When data is ASCII-encoded (DISP:HEAD:AFOR ASC):

<block_data>::={<point>,}...<point n><LF><^END>

<point>::=the y-axis values for the 1st through nth x-axis points (n is returned with the DISP:HEAD:POIN? query)

All y-axis values are returned in NRf format.

When data is binary-encoded (DISP:HEAD:AFOR FP32 or DISP:HEAD:AFOR FP64):

<block_data>::=#<byte><length_bytes>{<point>}...

<byte>::=one ASCII-encoded byte that specifies the number of length bytes to follow

<length_bytes>::=ASCII-encoded bytes that specify the number of data bytes to follow

<point>::=the y-axis values for the 1st through nth x-axis points (n is returned with DISP:HEAD:POIN?)

If DISP:HEAD:AFOR FP32 is specified, y-axis values are encoded as 32-bit binary floating point numbers. If DISP:HEAD:AFOR FP64 is specified, y-axis values are encoded as 64-bit binary floating point numbers.

Description:

This query dumps data from the specified display to the analyzer's output queue. Your controller can then read the data from the queue. The data returned by this query has already undergone a coordinate transform, so the y-axis values are in the current display units (returned from DISP:HEAD:YUN?).

The x-axis value for a given point is implied by the order of the points. DISP:HEAD:XOR is the x-axis value for the first point. Add DISP:HEAD:XINC to the first point's *x*-axis value to get the value of the second point. Add DISP:HEAD:XINC to the second point's x-axis value to get the value of the third point and so on.

DISP:GRATicule[?] command/query

Overlapped: no
Delayed result: no
Pass control required: no
Power-up state: 1

Example Statements:
```
OUTPUT 711;"Disp:Grat 0"
OUTPUT 711;"DISPLAY:B:GRATICULE ON"
OUTPUT 711;"DISP2:GRAT?"
```

Command Syntax: DISPlay[<spec>]:GRATicule<sp>{OFF|ON|0|1}

<spec>::=~:A~|:B|1|2

Query Syntax: DISPlay[<spec>]:GRATicule?

Returned Format: {0|1}<LF><^END>

Description:

Each display's graticule lines (or trace grid) can be turned on and off with this command. When a grid is turned off (DISP:GRAT OFF), it is not displayed, plotted, or printed.

The query returns 0 if the specified display's graticule is off, 1 if it is on.

DISP:HEADer selector

Description:

This command only selects the DISP:HEAD subsystem. Queries in this subsystem are used to determine characteristics of the data returned by the DISP:DATA query. Sending DISP:HEAD alone does nothing.

DISP:HEAD:AFORmat[?] command/query

Overlapped: no
Delayed result: no
Pass control required: no
Power-up state: ASC

Example Statements:

```
OUTPUT 711;"DISP:B:HEAD:AFOR FP64"
OUTPUT 711;"DISPLAY:HEADER:AFORMAT ASCII"
OUTPUT 711;"Disp:Head:Afor?"
```

Command Syntax: DISPlay[<spec>]:HEADer:AFORmat<sp>{ASCii|FP32|FP64}

<spec>::=~:A~|:B|1|2

Query Syntax: DISPlay[<spec>]:HEADer:AFORmat?

Returned Format: {ASC|FP32|FP64}<LF><^END>

Description:

Display data can either be ASCII-encoded or binary-encoded when it is dumped to the analyzer's output queue using the DISP:DATA query. This command lets you specify how the display data should be encoded.

NOTE Data encoding must be the same for both displays at any given time. So regardless of the display you specify when you send this command, encoding for both will be changed.

When ASC is selected, data is sent as a series of y-axis values separated by commas. The values are ASCII-encoded and are formatted as NRf decimal numbers.

FP32 and FP64 both specify binary encoding. When FP32 is selected, data is sent as a series of y-axis values within a definite length block. The values are encoded as 32-bit binary floating point numbers. When FP64 is selected, data is also sent as a series of y-axis values within a definite length block. However, the values are encoded as 64-bit binary floating point numbers.

For more information on data encoding and data transfer formats, see Chapter 4, "Transferring Data."

The query returns ASC, FP32, or FP64, depending on the option currently specified.

DISP:HEAD:NAME? query

Overlapped: no
Delayed result: no
Pass control required: no
Power-up state: "Spectrum Chan 1"(display A)
"Time Chan 1" (display B)

Example Statement: `OUTPUT 711;"DISP1:HEAD:NAME?"`

Query Syntax: DISPlay[<spec>]:HEADer:NAME?

<spec>::=~:A~|:B|1|2

Returned Format: "<trace_name>"<LF><^END>

<trace_name>::=0 to 30 printable ASCII characters

Description:

This query returns the name of the specified display. When looking at the analyzer's screen, you will see the name in the lower-left corner of the specified display.

You can change the name with the TRAC:TITL command.

DISP:HEAD:POINts? query

Overlapped: no
Delayed result: no
Pass control required: no
Power-up state: 401 (display A)
1024 (display B)

Example Statement: `OUTPUT 711;"DISP2:HEAD:POIN?"`

Query Syntax: DISPlay[<spec>]:HEADer:POINts?

<spec>::=~:A~|:B|1|2

Returned Format: <value><LF><^END>

<value>::=an integer (NR1 format)

Description:

A display's x-axis is divided into discrete points. Use this query to determine how many discrete points there are along the specified display's x-axis. This is the number of points that will be dumped to the analyzer's output queue when you send the DISP:DATA query.

DISP:HEAD:PREamble? query

Overlapped: no
Delayed result: no
Pass control required: no
Power-up state: variable

Example Statement: `OUTPUT 711;"DISP:HEAD:PRE?"`

Query Syntax: DISPlay[<spec>]:HEADer:PREamble?

<spec>::=~:A~|:B|1|2

Returned Format: <points>,<x_per_point>,<x_origin>,<x_increment>,
<y_per_point>,<y_origin>,<y_increment><LF><^END>

<points>::=number of discrete points on the display's x-axis (same as returned with DISP:HEAD:POIN?)

<x_per_point>::=number of x-axis values per point (same as returned with DISP:HEAD:XPO?)

<x_origin>::=x-axis value of the first point (same as returned with DISP:HEAD:XOR?)

<x_increment>::=increment between x-axis points (same as returned with DISP:HEAD:XINC?)

<y_per_point>::=number of y-axis values per point (same as returned with DISP:HEAD:YPO?)

<y_origin>::=y-axis value of the lowest point on the specified trace (same as returned with DISP:HEAD:YOR?)

<y_increment>::=optimum y-axis value per division (same as returned with DISP:HEAD:YINC?)

<points>, <x_per_point>, and <y_per_point> are integers (NR1 format). All other values are decimal numbers (NR2 or NR3 format).

Description:

This query returns seven pieces of information separated by commas. The information is useful for setting up an array to receive display data (returned from DISP:DATA?).

NOTE As the Returned Format indicates, each piece of information can be returned separately in response to its own query.

The <points>, <x_per_point>, and <y_per_point> values are used together to tell you how many values you must read after sending the DISP:DATA query. The formula is:

# of values to read = <points>×(<x_per_point+<y_per_point>)

The analyzer does not return x-axis values for each data point. Instead, it provides <x_origin> and <x_increment> values so you can assign an x-axis value to each returned point. <x_origin> is the x-axis value for the first point. Add <x_increment> to the first point's x-axis value to get the value of the second point. Add <x_increment> to the second point's x-axis value to get the value of the third point and so on.

The values returned in <y_origin> and <y_increment> should be ignored when the value of <y_points> is something other than 0 (zero).

DISP:HEAD:XINCrement? query

Overlapped: no
Delayed result: no
Pass control required: no
Power-up state: 256 (display A)
3.81E-6 (display B)

Example Statement: `OUTPUT 711;"DISP:A:HEAD:XINC?"`

Query Syntax: DISPlay[<spec>]:HEADer:XINCrement?

<spec>::=~:A~|:B|1|2

Returned Format: <value><LF><^END>

<value>::=a decimal number (NRf format)

Description:

This query returns the increment between x-axis values on the specified display. The value is only valid when the DISP:HEAD:XPO? response is 0.

DISP:HEAD:XINC and DISP:HEAD:XOR are used together to assign x-axis values to the points returned by the DISP:DATA query. DISP:HEAD:XOR is the x-axis value for the first point. Add DISP:HEAD:XINC to the first point's x-axis value to get the value of the second point. Add DISP:HEAD:XINC to the second point's x-axis value to get the value of the third point and so on.

Use DISP:HEAD:XUN? to determine units for the DISP:HEAD:XINC value.

DISP:HEAD:XNAMe? query

Overlapped: no
Delayed result: no
Pass control required: no
Power-up state: "Frequency" (display A)
"Time" (display B)

Example Statement: `OUTPUT 711;"DISP1:HEAD:XNAM?"`

Query Syntax: DISPlay[<spec>]:HEADer:XNAMe?

<spec>::=~:A~|:B|1|2

Returned Format: "{Frequency|Time}"<LF><^END>

Description:

This query returns the name of the specified display's x-axis. The name tells you whether the displayed data is in the frequency or the time domain.

DISP:HEAD:XORigin? query

Overlapped: no
Delayed result: no
Pass control required: no
Power-up state: 0

Example Statement: `OUTPUT 711;"DISP:B:HEAD:XOR?"`

Query Syntax: DISPlay[<spec>]:HEADer:XORigin?

<spec>::=~:A~|:B|1|2

Returned Format: <value><LF><^END>

<value>::=a decimal number (NRf format)

Description:

This query returns the x value of the specified display's first x-axis point. The value is only valid when the DISP:HEAD:XPO? response is 0. The analyzer always returns 0 when DISP:HEAD:XPO? is sent.

DISP:HEAD:XOR and DISP:HEAD:XINC are used together to assign x-axis values to the points returned by DISP:DATA?. See DISP:HEAD:XINC for more information.

Use DISP:HEAD:XUN? to determine units for the DISP:HEAD:XOR value.

DISP:HEAD:XPOints? query

Overlapped: no
Delayed result: no
Pass control required: no
Power-up state: 0

Example Statement: `OUTPUT 711;"DISP2:HEAD:XPO?"`

Query Syntax: DISPlay[<spec>]:HEADer:XPOints?

<spec>::=~:A~|:B|1|2

Returned Format: 0<LF><^END>

Description:

The DISP:DATA query returns data from the specified display as a series of data points. The DISP:HEAD:XPO query tells you how many x-axis values will be returned with each point.

Since each data point can only be made up of y-axis values, the DISP:HEAD:XPO query always returns 0. You can calculate the x-axis values for each point using the values returned by the DISP:HEAD:XINC and DISP:HEAD:XOR queries.

DISP:HEAD:XUNits? query

Overlapped: no
Delayed result: no
Pass control required: no
Power-up state: "HZ" (display A)
"S" (display B)

Example Statement: `OUTPUT 711;"DISP:A:HEAD:XUN?"`

Query Syntax: DISPlay[<spec>]:HEADer:XUNits?

<spec>::=~:A~|:B|1|2

Returned Format: "{HZ|S}"<LF><^END>

Description:

This query tells you what units apply to the DISP:HEAD:XINC and DISP:HEAD:XOR values.

DISP:HEAD:YINCrement? query

Overlapped: no
Delayed result: no
Pass control required: no
Power-up state: variable

Example Statement: `OUTPUT 711;"DISP1:HEAD:YINC?"`

Query Syntax: DISPlay[<spec>]:HEADer:YINCrement?

<spec>::=~:A~|:B|1|2

Returned Format: <value><LF><^END>

<value>::=a decimal number (NRf format)

Description:

This query returns the optimum y-axis value per division for the specified trace. The value returned is the result of the following calculation:

$$(Ymax - Ymin)/8$$

Where:

Ymax = the y-axis value of the highest point on the trace
Ymin = the y-axis value of the lowest point on the trace

The value is returned in the current y-axis units.

DISP:HEAD:YNAMe? query

Overlapped: no
Delayed result: no
Pass control required: no
Power-up state: "LogMag" (display A)
"Real" (display B)

Example Statement: `OUTPUT 711;"DISP:B:HEAD:YNAM?"`

Query Syntax: DISPlay[<spec>]:HEADer:YNAMe?

<spec>::=~:A~|:B|1|2

Returned Format: "{Delay|Imag|LinMag|LogMag|Phase|Real}"<LF><^END>

Description:

This query returns the name of the specified display's y-axis. The name tells you what kind of coordinates are being used to display the data. (Coordinates are referred to as Trace Type on the analyzer's front panel.)

DISP:HEAD:YORigin? query

Overlapped: no
Delayed result: no
Pass control required: no
Power-up state: variable

Example Statement: `OUTPUT 711;"DISP2:HEAD:YOR?"`

Query Syntax: DISPlay[<spec>]:HEADer:YORigin?

<spec>::=~:A~ |:B|1|2

Returned Format: <value><LF><^END>

<value>::=a decimal number (NRf format)

Description:

This query returns the y-axis value of the lowest point on the specified trace.

DISP:HEAD:YPOints? query

Overlapped: no
Delayed result: no
Pass control required: no
Power-up state: 1

Example Statement: `OUTPUT 711;"DISP:HEAD:YPO?"`

Query Syntax: DISPlay[<spec>]:HEADer:YPOints?

<spec>::=~:A~ |:B|1|2

Returned Format: 1<LF><^END>

Description:

The DISP:DATA query returns data from the specified display as a series of data points. The DISP:HEAD:YPO query tells you how many y-axis values will be returned with each point.

The analyzer always returns 1 in response to this query. This means that each point returned by DISP:DATA? will consist of one y-axis value.

DISP:HEAD:YUNits? query

Overlapped: no
Delayed result: no
Pass control required: no
Power-up state: "DBVRMS" (display A)
"V" (display B)

Example Statement: `OUTPUT 711;"DISP2:HEAD:YUN?"`

Query Syntax: DISPlay[<spec>]:HEADer:YUNits?

<spec>::=~:A~|:B|1|2

Returned Format: "[<unit>]"<LF><^END>

<unit>::=V|V2|VRMS|VRMS2|DB|DBM|
DBVRMS|DBVPK|DEG|RAD|V/RTHZ|
VRMS/RTHZ|V2/HZ|VRMS2/HZ|
DBVRMS/HZ|DBVPK/HZ|DBM/HZ|S

Description:

This query tells you what unit applies to the y-axis values returned from the DISP:DATA query.

NOTE Not listed in Returned Format are the many special units that can result from math operations or the application of engineering units. However, such units are also valid responses.

DISP:LIMit[?] command/query

Description:

Disp:LIM is functionally equivalent to Disp:LIM:TABL. See the latter command for more details.

DISP:LIM:BEEPer[?] command/query

Overlapped: no
Delayed result: no
Pass control required: no
Power-up state: 0

Example Statements:

```
OUTPUT 711;"disp2:lim:beep off"
OUTPUT 711;"Display:A:Limit:Beeper On"
OUTPUT 711;"DISP:LIM:BEEP?"
```

Command Syntax: DISPlay[<spec>]:LIMit:BEEPer<sp>{OFF|ON|0|1}

<spec>::=~:A~|:B|1|2

Query Syntax: DISPlay[<spec>]:LIMit:BEEPer?

Returned Format: {0|1}<LF><^END>

Description:

This command enables and disables the limit-test beeper. When the beeper is enabled (DISP:LIM:BEEP ON) and the limit test fails, the analyzer beeps.

The system beeper must also be enabled (SYST:BEEP ON) if you want the analyzer to beep.

The query returns 0 if the limit beeper is off, 1 if it is on.

DISP:LIM:FAIL? query

Description:

Disp:LIM:FAIL is functionally equivalent to Disp:LIM:FAIL:DATA. See the latter query for more details.

DISP:LIM:FAIL[:DATA]? query

Overlapped: no
Delayed result: no
Pass control required: no
Power-up state: 0

Example Statement: `OUTPUT 711;"DISP:B:LIM:FAIL?"`

Query Syntax: DISPlay[<spec>]:LIMit:FAIL[:DATA]?

<spec>::=~:A~|:B|1|2

Returned Format: <block_data>

<block_data> takes one of two forms, depending on whether the data is ASCII-encoded or binary-encoded. When data is ASCII-encoded (DISP:LIM:FAIL:HEAD:AFOR ASC):

<block_data>::={<point>,}...<point n><LF><^END>

<point>::=<x_value>,<y_value>,<y_limit>,<y_flag>

These values are returned for the 1st through nth points (n is returned with DISP:LIM:FAIL:HEAD:POIN?)

All values are returned in the NRf format and are separated by commas.

When data is binary-encoded, (DISP:LIM:FAIL:HEAD:AFOR FP32 or DISP:LIM:FAIL:HEAD:AFOR FP64):

<block_data>::=#<byte><length_bytes>{<point>}...

<byte>::=one ASCII-encoded byte that specifies the number of length bytes to follow

<length_bytes>::=ASCII-encoded bytes that specify the number of data bytes to follow

<point>::=<x_value><y_value><y_limit><y_flag>

These values are returned for the 1st through nth points (n is returned with DISP:LIM:FAIL:HEAD:POIN?)

All values are returned as either 32-bit or 64-bit binary floating point numbers, depending on the setting of DISP:LIM:FAIL:HEAD:AFOR.

Description:

When a limit table is coupled to a display that has limit testing enabled, the data in that display is tested against limits specified in the table. Limits may be set for some or all of the displayed data. This query responds with those points of the specified data that failed when tested against the limits. Each point consists of four values, which are defined as follows:

<x_value>::=x-axis value of the failed point

<y_value>::=y-axis value of the failed point

<y_limit>::=y limit specified for the failed point

<y_flag>::=fail flag (0=passed, 1=failed min. limit, 2=failed max. limit)

Limit tables are defined with the LIM:TABL:DATA command. They are assigned to a display using the DISP:LIM:TABL command.

DISP:LIM:FAIL:HEADer — selector

Description:

This command only selects the DISP:LIM:FAIL:HEAD subsystem. Sending DISP:LIM:FAIL:HEAD alone does nothing.

DISP:LIM:FAIL:HEAD:AFORmat[?] — command/query

Overlapped: no
Delayed result: no
Pass control required: no
Power-up state: ASC

Example Statements:

```
OUTPUT 711;"DISP:A:LIM:FAIL:HEAD:AFOR FP64"
OUTPUT 711;"DISPLAY:B:LIMIT:FAIL:HEADER:AFORMAT ASCII"
OUTPUT 711;"DISP:LIM:FAIL:HEAD:AFOR?"
```

Command Syntax: DISPlay[<spec>]:LIMit:FAIL:HEADer:AFORmat<sp>{ASCii|FP32|FP64}

<spec>::=~:A~|:B|1|2

Query Syntax: DISPlay[<spec>]:LIMit:FAIL:HEADer:AFORmat?

Returned Format: {ASC|FP32|FP64}<LF><^END>

Description:

Data returned in response to the DISP:LIM:FAIL:DATA query can be ASCII-encoded or binary-encoded. This command allows you to specify how each limit's data should be encoded.

When ASC is selected, data is sent as a series of values separated by commas. The values are ASCII-encoded and are formatted as NRf decimal numbers.

FP32 and FP64 both specify binary encoding. When FP32 is selected, data is sent as a series of values within a definite length block. The values are encoded as 32-bit binary floating point numbers. When FP64 is selected, data is also sent as a series of values within a definite length block. However, the values are encoded as 64-bit binary floating point numbers.

For more information on data encoding and data transfer formats, see Chapter 4, "Transferring Data."

The query returns ASC, FP32, or FP64, depending on the option currently specified.

DISP:LIM:FAIL:HEAD:POINts? query

Overlapped: no
Delayed result: no
Pass control required: no
Power-up state: 0

Example Statement:
```
OUTPUT 711;"DISP1:LIM:FAIL:HEAD:POIN?"
```

Query Syntax: DISPlay[<spec>]:LIMit:FAIL:HEADer:POINts?

<spec>::=~:A~|:B|1|2

Returned Format: <value><LF><^END>

<value>::=an integer (NR1 format)

Description:

This query tells you how many points in the specified display failed when tested against a limit table.

To define limit tables, use the LIM:TABL:DATA command. To assign limit tables to one of the displays, use the DISP:LIM:TABL command. To read the values of the failed points, use the DISP:LIM:FAIL:DATA query.

DISP:LIM:LINE[?] command/query

Overlapped: no
Delayed result: no
Pass control required: no
Power-up state: 0

Example Statements:
```
OUTPUT 711;"Disp:Lim:Line On"
OUTPUT 711;"DISPLAY1:LIMIT:LINE 0"
OUTPUT 711;"DISP:LIM:LINE?"
```

Command Syntax: DISPlay[<spec>]:LIMit:LINE<sp>{OFF|ON|0|1}

<spec>::=~:A~|:B|1|2

Query Syntax: DISPlay[<spec>]:LIMit:LINE?

Returned Format: {0|1}<LF><^END>

Description:

This command enables the specified display to show limit lines. These limit lines define the bounds within which you want the trace data to fall.

The query returns 0 if the specified display is not enabled to show limit lines, 1 if it is.

DISP:LIM:STATe[?] command/query

Overlapped: no
Delayed result: no
Pass control required: no
Power-up state: 0

Example Statements:

```
OUTPUT 711;"DISP:A:LIM:STAT OFF"
OUTPUT 711;"Display2:Limit:State 1"
OUTPUT 711;"disp:b:lim:stat?"
```

Command Syntax: DISPlay[<spec>]:LIMit:STATe<sp>{OFF|ON|0|1}

<spec>::=~:A~|:B|1|2

Query Syntax: DISPlay[<spec>]:LIMit:STATe?

Returned Format: {0|1}<LF><^END>

Description:

This command enables limit testing for the specified display. ON and 1 enable limit testing; OFF and 0 disable limit testing.

While limit testing is enabled for a particular display, the data on that display is tested against the limits each time the display is updated. When display A data fails the test, bit 8 of the Data Integrity Condition register is set. When display B data fails the test, bit 9 of the Data Integrity Condition register is set.

The query returns 0 if limit testing is not enabled for the specified display, 1 if it is.

DISP:LIM[:TABLe][?] command/query

Overlapped: no
Delayed result: no
Pass control required: no
Power-up state: 1 (display A)
2 (display B)

Example Statements:
```
OUTPUT 711;"Disp2:Lim 1"
OUTPUT 711;"DISPLAY:A:LIMIT:TABLE 8"
OUTPUT 711;"DISP:LIM?"
```

Command Syntax: DISPlay[<spec>]:LIMit[:TABLe]<sp><table_number>

<spec>::=~:A~|:B|1|2

<table_number>::=a single integer 1 through 8 (NRf format)

Query Syntax: DISPlay[<spec>]:LIMit[:TABLe]?

Returned Format: <value><LF><^END>

<value>::=an integer (NR1 format)

Description:

This command allows you to couple one of the eight limit tables to the specified display.

If limit testing is enabled, the displayed data is automatically tested against the limit table you specify with this command. The data is tested each time the display is updated.

The query response indicates which limit table is coupled to the specified trace.

DISP:LIM:TEST? query

Description:

Disp:LIM:TEST is functionally equivalent to Disp:LIM:TEST:DATA. See the latter query for more details.

DISP:LIM:TEST[:DATA]? query

Overlapped: no
Delayed result: no
Pass control required: no
Power-up state: 0

Example Statement: `OUTPUT 711;"DISP:A:LIM:TEST?"`

Query Syntax: DISPlay[<spec>]:LIMit:TEST[:DATA]?

<spec>::=~:A~|:B|1|2

Returned Format: <block_data>

<block_data> takes one of two forms, depending on whether the data is ASCII-encoded or binary-encoded. When data is ASCII-encoded (DISP:LIM:FAIL:HEAD:AFOR ASC):

<block_data>::={<point>,}...<point n><LF><^END>

<point>::=<x_value>,<y_value>,<y_limit>,<y_flag>

These values are returned for the 1st through nth points (n is returned with DISP:LIM:FAIL:HEAD:POIN?)

All values are returned in the NRf format and are separated by commas.

When data is binary-encoded, (DISP:LIM:FAIL:HEAD:AFOR FP32 or DISP:LIM:FAIL:HEAD:AFOR FP64):

<block_data>::=#<byte><length_bytes>{<point>}...

<byte> ::=one ASCII-encoded byte that specifies the number of length bytes to follow

<length_bytes>::=ASCII-encoded bytes that specify the number of data bytes to follow

<point>::=<x_value><y_value><y_limit><y_flag>

These values are returned for the 1st through nth points (n is returned with DISP:LIM:FAIL:HEAD:POIN?)

All values are returned as either 32-bit or 64-bit binary floating point numbers, depending on the setting of DISP:LIM:FAIL:HEAD:AFOR.

Description:

When a limit table is coupled to a display that has limit testing enabled, the data in that display is tested against limits specified in the table. Limits may be set for some or all of the displayed data. This query responds with all points of the specified data that were tested against limits, even if they did not fail those limits. Each point consists four values, which are defined as follows:

<x_value>::=x-axis value of the tested point

<y_value>::=y-axis value of the tested point

<y_limit>::=y limit specified for the tested point

<y_flag>::=fail flag (0=passed, 1=failed min. limit, 2=failed max. limit)

Limit tables are defined the LIM:TABL:DATA command. They are assigned to a display using the DISP:LIM:TABL command.

DISP:LIM:TEST:HEADer selector

Description:

This command only selects the DISP:LIM:TEST:HEAD subsystem. Sending DISP:LIM:TEST:HEAD alone does nothing.

DISP:LIM:TEST:HEAD:AFORmat[?] command/query

Overlapped: no
Delayed result: no
Pass control required: no
Power-up state: ASC

Example Statements:

```
OUTPUT 711;"DISP1:LIM:TEST:HEAD:AFOR ASCii"
OUTPUT 711;"display:b:limit:test:header:aformat fp64"
OUTPUT 711;"DISP:LIM:TEST:HEAD:AFOR?"
```

Command Syntax: DISPlay[<spec>]:LIMit:TEST:HEADer:AFORmat<sp>
{ASCii|FP32|FP64}

<spec>::=~:A~|:B|1|2

Query Syntax: DISPlay[<spec>]:LIMit:TEST:HEADer:AFORmat?

Returned Format: {ASC|FP32|FP64}<LF><^END>

Description:

Data returned in response to the DISP:LIM:TEST:DATA query can be ASCII-encoded or binary-encoded. This command allows you to specify how each display's data should be encoded.

When ASC is selected, data is sent as a series of values separated by commas. The values are ASCII-encoded and are formatted as NRf decimal numbers.

FP32 and FP64 both specify binary encoding. When FP32 is selected, data is sent as a series of values within a definite length block. The values are encoded as 32-bit binary floating point numbers. When FP64 is selected, data is also sent as a series of values within a definite length block. However, the values are encoded as 64-bit binary floating point numbers.

For more information on data encoding and data transfer formats, see Chapter 4, "Transferring Data."

The query returns ASC, FP32, or FP64, depending on the option currently specified.

DISP:LIM:TEST:HEAD:POINts? query

Overlapped: no
Delayed result: no
Pass control required: no
Power-up state: 0

Example Statement: `OUTPUT 711;"DISP:LIM:TEST:HEAD:POIN?"`

Query Syntax: DISPlay[<spec>]:LIMit:TEST:HEADer:POINts?

<spec>::=~:A~|:B|1|2

Returned Format: <value><LF><^END>

<value>::=an integer (NR1 format)

Description:

This query tells you how many points in the specified display were tested against a limit table.

To define limit tables, use the LIM:TABL:DATA command. To assign limit tables to one of the displays, use the DISP:LIM:TABL command. To read the values of the tested points, use the DISP:LIM:FAIL:DATA query.

DISP:X selector

Description:

This command only selects the DISP:X subsystem. Sending DISP:X alone does nothing.

DISP:X:APERture[?] command/query

Overlapped: no
Delayed result: no
Pass control required: no
Power-up state: 0.005

Example Statements:

```
OUTPUT 711;"DISP:B:X:APER 0.01"
OUTPUT 711;"DISPLAY1:X:APERTURE 16PCT"
OUTPUT 711;"Disp:B:X:Aper?"
```

Command Syntax: DISPlay[<spec>]:X:APERture<sp>{{<percent>PCT}|<fraction>}

<spec>::=~:A~|:B|1|2

<percent>::=.5|1|2|4|8|16

<fraction>::=0.005|0.01|0.02|0.04|0.08|0.16

Query Syntax: DISPlay[<spec>]:X:APERture?

Returned Format: <fraction><LF><^END>

<fraction>::=a decimal number (NR2 format)

Description:

When group delay coordinates are used (DISP:Y:AXIS GDEL), you must select a phase-smoothing aperture. The greater the aperture you select, the greater will be the smoothing effect on the displayed data. This command allows you to select an aperture for the specified display.

The aperture is entered as a percentage or as a fraction of the current frequency span. DISP:X:APER 0.01 is the same as DISP:X:APER 1PCT. In either case, the value you send is rounded to the nearest allowable percentage.

The query response indicates which aperture is currently selected for the specified trace. The value is returned in the fractional form.

DISP:X:SPACing[?] command/query

Overlapped: no
Delayed result: no
Pass control required: no
Power-up state: LIN

Example Statements:

```
OUTPUT 711;"Disp2:X:Spac Lin"
OUTPUT 711;"DISPLAY1:X:SPACING LOGARITHMIC"
OUTPUT 711;"DISP:X:SPAC?"
```

Command Syntax: DISPlay[<spec>]:X:SPACing<sp>{LINear|LOGarithmic}

<spec>::=~:A~|:B|1|2

Query Syntax: DISPlay[<spec>]:X:SPACing?

Returned Format: {LIN|LOG}<LF><^END>

Description:

Use this command to specify whether the spacing of data points along the x-axis should be linear or logarithmic.

The query returns LIN if linear spacing is selected and LOG if logarithmic spacing is selected for the specified display.

DISP[:Y] selector

Description:

This command only selects the DISP:Y subsystem. Sending DISP:Y alone does nothing.

DISP[:Y]:AXIS[?] command/query

Overlapped: no
Delayed result: no
Pass control required: no
Power-up state: LOGM (display A)
REAL (display B)

Example Statements:

```
OUTPUT 711;"DISP:AXIS LINM"
OUTPUT 711;"Display2:Y:Axis Magnitude"
OUTPUT 711;"disp:axis?"
```

Command Syntax: DISPlay[<spec>][:Y]:AXIS<sp><axis>

<spec>::=~:A~|:B|1|2

<axis>::=GDELay|IMAGinary|LINMagnitude|LOGMagnitude|PHASe|REAL

Query Syntax: DISPlay[<spec>][:Y]:AXIS?

Returned Format: {GDEL|IMAG|LINM|LOGM|PHAS|REAL}<LF><^END>

Description:

This command lets you specify the coordinate system to be used for the specified display. (Coordinate systems are referred to as Trace Types on the analyzer's front panel.)

DISP:Y:AXIS GDEL specifies the group delay coordinate system, which uses time on the y-axis and frequency on the x-axis. Group delay is related to phase, but shows phase delays in time rather than degrees of phase shift. The analyzer uses a smoothing aperture to define the resolution of the group delay display. This coordinate system is not allowed for time records. See DISP:X:APER for more information.

DISP:Y:AXIS IMAG specifies the imaginary coordinate system, which uses imaginary numbers for the y-axis and frequency or time for the x-axis. This coordinate system shows the imaginary component of complex data at each point along the x-axis.. If the data is real rather than complex, a y value of 0 is displayed for all x-axis points.

DISP:Y:AXIS LINM specifies the linear magnitude coordinate system, which uses magnitude for the y-axis and frequency or time for the x-axis. In addition, the y-axis scale is spaced linearly. DISP:Y:AXIS LOGM specifies the logarithmic magnitude coordinate system, which also uses magnitude for the y-axis and frequency or time for the x-axis. However, the y-axis scale is spaced logarithmically.

DISP:Y:AXIS PHAS specifies the phase coordinate system, which uses phase for the y-axis and frequency or time for the x-axis.

DISP:Y:AXIS REAL specifies the real coordinate system, which uses real numbers for the y-axis and frequency or time for the x-axis. This coordinate system shows real data or the real component of complex data at each point along the x-axis.

The query response tells you which scaling system is currently selected.

DISP[:Y]:SCALe selector

Description:

This command only selects the DISP:Y:SCAL subsystem. Sending DISP:Y:SCAL alone does nothing.

DISP[:Y]:SCAL:AUTO selector

Description:

This command only selects the DISP:Y:SCAL:AUTO subsystem. Sending DISP:Y:SCAL:AUTO alone does nothing.

DISP[:Y]:SCAL:AUTO:SINGle command

Overlapped: no
Delayed result: no
Pass control required: no
Power-up state: not applicable

Example Statements:

```
OUTPUT 711;"DISP:A:SCAL:AUTO:SING"
OUTPUT 711;"Display2:Y:Scale:Auto:Single"
```

Command Syntax: DISPlay[<spec>][:Y]:SCALe:AUTO:SINGle

<spec>::=~:A~|:B|1|2

Description:

This command performs a single autoscale on the specified display. This optimizes y-axis scaling for that display.

DISP[:Y]:SCAL:CENTer[?] command/query

Overlapped: no
Delayed result: no
Pass control required: no
Power-up state: variable

Example Statements:
```
OUTPUT 711;"disp1:scal:cent 5v"
OUTPUT 711;"DISPLAY:B:Y:SCALE:CENTER -40 DBM"
OUTPUT 711;"Disp:B:Scal:Cent?"
```

Command Syntax: DISPlay[<spec>][:Y]:SCALe:CENTer<sp><value>[<unit>]

<spec>::=~:A~|:B|1|2

<value>::=a decimal number (NRf format)

<unit>options are listed in Appendix A.

Query Syntax: DISPlay[<spec>][:Y]:SCALe:CENTer?

Returned Format: <value><LF><^END>

<value>::=a decimal number (NRf format)

Description:

This command allows you to define the center of a display's vertical scale. Changing the vertical-per-division value (DISP:Y:SCAL:DIV) after using this command will alter the top and bottom points of the display while keeping the center point fixed.

The unit you can send with this command depends on two things:

- The measurement data being displayed
- The coordinate system (also called trace type) being used to display the measurement data

Send the TRAC:RES query to determine which measurement data is being displayed and the DISP:Y:AXIS query to determine which trace type is being used. You can then refer to Appendix A to determine which units you can send with this command.

NOTE If you do not include a <unit> specifier when you send this command, the analyzer assumes a default unit. This default unit is not necessarily the current unit used for the vertical axis. Default units are specified in Appendix A.

The query returns the center point of the display's vertical scale. Note that only a value is returned; units are not appended. Units are returned by the DISP:Y:SCAL:UNIT query.

DISP[:Y]:SCAL:DIVision[?] command/query

Overlapped: no
Delayed result: no
Pass control required: no
Power-up state: variable

Example Statements:

```
OUTPUT 711;"DISP:SCAL:DIV 5"
OUTPUT 711;"DISPLAY1:Y:SCALE:DIVISION 10"
OUTPUT 711;"disp1:scal:div?"
```

Command Syntax: DISPlay[<spec>][:Y]:SCALe:DIVision<sp><value>[<unit>]

<spec>::=~:A~|:B|1|2

<value>::=any decimal number x, where .001 ≤ x ≤ 100 (when the display units are referenced to dB)

any decimal number x, where 1 E-36 ≤ x ≤ 1 E36 (when the display units are not referenced to dB)

<unit>options are listed in Appendix A.

Query Syntax: DISPlay[<spec>][:Y]:SCALe:DIVision?

Returned Format: <value><LF><^END>

<value>::=a decimal number (NRf format)

Description:

Graticule lines divide a display's vertical axis into eight divisions. Use this command to define the increment between graticule lines on the specified display's vertical axis.

The unit you can send with this command depends on two things:

- The measurement data being displayed
- The coordinate system (also called trace type) being used to display the measurement data

Send the TRAC:RES query to determine which measurement data is being displayed and the DISP:Y:AXIS query to determine which trace type is being used. You can then refer to Appendix A to determine which units you can send with this command.

NOTE If you do not include a <unit> specifier when you send this command, the analyzer assumes a default unit. This default unit is not necessarily the current unit used for the vertical axis. Default units are specified in Appendix A.

The query returns the current increment between specified display's graticule lines. Note that only a value is returned; units are not appended.

DISP[:Y]:SCAL:REFerence[?] command/query

Overlapped: no
Delayed result: no
Pass control required: no
Power-up state: INP

Example Statements:
```
OUTPUT 711;"DISP:A:SCAL:REF CENT"
OUTPUT 711;"Display:B:Y:Scale:Reference Start"
OUTPUT 711;"DISP2:SCAL:REF?"
```

Command Syntax: DISPlay[<spec>][:Y]:SCALe:REFerence<sp>
{CENTer|STARt|STOP|INPut}

<spec>::=~:A~|:B|1|2

Query Syntax: DISPlay[<spec>][:Y]:SCALe:REFerence?

Returned Format: {CENT|STAR|STOP|INP}<LF><^END>

Description:

This command lets you use one of three parameters to select the top, center, or bottom of the specified display as a vertical axis reference point. STOP selects the top, CENT selects the center, and STAR selects the bottom. The reference point selected with this command remains fixed when the vertical-per-division value (DISP:Y:SCAL:DIV) is changed.

A fourth parameter, INP, enables automatic reference tracking. Automatic reference tracking selects vertical scaling values based on the input range of the channel supplying the measurement data.

Your selection of trace type and measurement data affects reference level tracking. When the logarithmic magnitude trace type is selected, the top reference is kept at the input range. When the linear magnitude trace type is selected, the bottom reference is kept at zero. When the real or imaginary trace types are selected, the center reference is set to 0 (zero). In addition, when linear magnitude, real, or imaginary trace types are selected, the vertical-per-division value is changed so that the top reference is ≥ the input range. (See the DISP:Y:AXIS command for information on specifying the trace type.)

Reference level tracking is not allowed for the phase (DISP:Y:AXIS PHAS) and group delay (DISP:Y:AXIS GDEL) trace types. It is also not allowed for frequency response, coherence, and user-defined measurement data. (See TRAC:RES for information on measurement data).

The vertical axis reference point is changed and automatic reference tracking is disabled by the following commands: DISP:Y:SCAL:CENT, DISP:Y:SCAL:STAR, and DISP:Y:SCAL:STOP. Reference level tracking is also disabled by these commands: DISP:Y:SCAL:AUTO:SING and DISP:Y:SCAL:DIV. (When DISP:Y:AXIS is LOGM, DISP:Y:SCAL:DIV might not disable reference level tracking.)

This command does not allow you to specify a value for the reference point. This must be done with the DISP:Y:SCAL:CENT, DISP:Y:SCAL:STAR, or DISP:Y:SCAL:STOP command.

The query response tells you what kind of vertical-axis scaling is currently selected.

DISP[:Y]:SCAL:STARt[?] command/query

Overlapped: no
Delayed result: no
Pass control required: no
Power-up state: variable

Example Statements:
```
OUTPUT 711;"Disp:Scal:Star 10"
OUTPUT 711;"display2:Y:scale:start -40dBVrms"
OUTPUT 711;"DISP:A:SCAL:STAR?"
```

Command Syntax: DISPlay[<spec>][:Y]:SCALe:STARt<sp><value>[<unit>]

<spec>::=~:A~|:B|1|2

<value>::=a decimal number (NRf format)

<unit>options are listed in Appendix A

Query Syntax: DISPlay[<spec>][:Y]:SCALe:STARt?

Returned Format: <value><LF><^END>

<value>::=a decimal number (NRf format)

Description:

This command allows you to define the bottom of a display's vertical scale. Changing the vertical-per-division value (DISP:Y:SCAL:DIV) after using this command will alter the top and center points of the display while keeping the bottom point fixed.

The unit you can send with this command depends on two things:

- The measurement data being displayed
- The coordinate system (also called trace type) being used to display the measurement data

Send the TRAC:RES query to determine which measurement data is being displayed and the DISP:Y:AXIS query to determine which trace type is being used. You can then refer to Appendix A to determine which units you can send with this command.

NOTE If you do not include a <unit> specifier when you send this command, the analyzer assumes a default unit. This default unit is not necessarily the current unit used for the vertical axis. Default units are specified in Appendix A.

The query returns the bottom point of the display's vertical scale. Note that only a value is returned; units are not appended. Units are returned by the DISP:Y:SCAL:UNIT query.

DISP[:Y]:SCAL:STOP[?] command/query

Overlapped: no
Delayed result: no
Pass control required: no
Power-up state: variable

Example Statements:
```
OUTPUT 711;"DISP:A:SCAL:STOP 1"
OUTPUT 711;"display2:Y:scale:stop 10dbvrms"
OUTPUT 711;"Disp:Scal:Stop?"
```

Command Syntax: DISPlay[<spec>][:Y]:SCALe:STOP<sp><value>[<unit>]

<spec>::=~:A~|:B|1|2

<value>::=a decimal number (NRf format)

<unit>options are listed in Appendix A

Query Syntax: DISPlay[<spec>][:Y]:SCALe:STOP?

Returned Format: <value><LF><^END>

<value>::=a decimal number (NRf format)

Description:

This command allows you to define the top of a display's vertical scale. Changing the vertical-per-division value (DISP:Y:SCAL:DIV) after using this command will alter the center and bottom points of the display while keeping the top point fixed.

The unit you can send with this command depends on two things:

- The measurement data being displayed
- The coordinate system (also called trace type) being used to display the measurement data

Send the TRAC:RES query to determine which measurement data is being displayed and the DISP:Y:AXIS query to determine which trace type is being used. You can then refer to Appendix A to determine which units you can send with this command.

NOTE If you do not include a <unit> specifier when you send this command, the analyzer assumes a default unit. This default unit is not necessarily the current unit used for the vertical axis. Default units are specified in Appendix A.

The query returns the top point of the display's vertical scale. Note that only a value is returned; units are not appended. Units are returned by the DISP:Y:SCAL:UNIT query.

DISP[:Y]:SCAL:UNITs[?] command/query

Overlapped: no
Delayed result: no
Pass control required: no
Power-up state: "DBVRMS" (display A)
"V" (display B)

Example Statements:

```
OUTPUT 711;"Disp:Scal:Unit ""Deg"""
OUTPUT 711;"DISPLAY2:Y:SCALE:UNITS 'DBVPK'"
OUTPUT 711;"disp1:scal:unit?"
```

Command Syntax: DISPlay[<spec>][:Y]:SCALe:UNITs<sp>{'|"}<unit>{'|"}

<spec>::=~:A~ |:B|1|2

<unit>options are listed in Appendix A

Query Syntax: DISPlay[<spec>][:Y]:SCALe:UNITs?

Returned Format: "<unit>"<LF><^END>

Description:

Use this command to select a unit for the specified display's y-axis.

The unit you can send with this command depends on two things:

- The measurement data being displayed
- The coordinate system (also called trace type) being used to display the measurement data

Send the TRAC:RES query to determine which measurement data is being displayed and the DISP:Y:AXIS query to determine which trace type is being used. You can then refer to Appendix A to determine which units you can send with this command.

The query returns the y-axis unit currently being used for the specified trace.

FREQuency subsystem

Description:

Commands in this subsystem are used to define the band of frequencies you want to analyze.

NOTE The amplitude accuracy of the HP 35660A is specified to a maximum of 102.4 kHz in the one-channel measurement mode and 51.2 kHz in the two-channel mode. However, commands in this subsystem allow you to define the band of frequencies in such a way that frequencies greater than the maximums are displayed. The amplitude accuracy of frequencies exceeding the specified maximums is not guaranteed.

FREQ:CENTer[?] command/query

Overlapped: no
Delayed result: yes
Pass control required: no
Power-up state: 51200

Example Statements:

```
OUTPUT 711;"FREQ:CENT 100"
OUTPUT 711;"FREQUENCY:CENTER 98KHZ"
OUTPUT 711;"Freq:Cent?"
```

Command Syntax: FREQuency:CENTer<sp><value>[<unit>]

<value>::=any number between 0 and max_disp_freq

Send numbers in the NRf format.

max_disp_freq::=115,000 for 1-channel measurements,
57,500 for 2-channel measurements

<unit>::=~HZ~ |KHZ

Query Syntax: FREQuency:CENTer?

Returned Format: <value><LF><^END>

<value>::=a decimal number (NRf format)

Description:

This command specifies the center of the band of frequencies you want to analyze. The values of FREQ:CENT and FREQ:SPAN completely define the band. When you send this command, the value of FREQ:STAR is automatically adjusted (if necessary) so that the following formula is true:

FREQ:STAR=FREQ:CENT–(FREQ:SPAN/2)

The same formula continues to adjust the value of FREQ:STAR each time FREQ:CENT or FREQ:SPAN is changed. This remains true until you explicitly set the value of FREQ:STAR or until you send FREQ:REF STAR, at which point FREQ:CENT becomes the value that is automatically adjusted.

You can either use numbers or one of three nonnumeric parameters to set the value of FREQ:CENT. The nonnumeric parameters are:

- UP – increases the current value of FREQ:CENT by the amount specified in FREQ:CENT:STEP
- DOWN – decreases the current value of FREQ:CENT by the amount specified in FREQ:CENT:STEP
- (MARK[:A|:B]:VAL) – sets FREQ:CENT to the frequency of the main marker, even when the marker reference is enabled

The query returns the current center of the band of frequencies being analyzed. The value is returned in Hz.

FREQ:CENT:STEP[?] command/query

Overlapped: no
Delayed result: yes
Pass control required: no
Power-up state: 2000

Example Statements:

```
OUTPUT 711;"Freq:Cent:Step 5"
OUTPUT 711;"frequency:center:step 2khz"
OUTPUT 711;"FREQ:CENT:STEP?"
```

Command Syntax: FREQuency:CENTer:STEP<sp><value>[<unit>]

<value>::=any x, where 0 ≤ x ≤ 51.2 kHz for a one-channel measurement

any x, where 0 ≤ x ≤ 25.6 kHz for a two-channel measurement

<unit>::=~HZ~ |KHZ

Query Syntax: FREQuency:CENTer:STEP?

Returned Format: <value><LF><^END>

<value>::=a decimal number (NRf format)

Description:

FREQ:CENT and FREQ:STAR can both be increased or decreased by a certain amount. The amount, called a step, is defined by this command.

You can either use a number or the parameter (MARK[:A|:B]:VAL) to set the step. (MARK:VAL) sets the step to one of two values depending on the marker mode selected:

- When MARK:X:MODE is NORM, (MARK:VAL) sets the step to the value of the main marker.
- When MARK:X:MODE is DELT, (MARK:VAL) sets the step to the difference between the marker reference value and the main marker value.

The query returns the step currently specified. The value is returned in Hz.

FREQ:REFerence commands/query

Overlapped: no
Delayed Result: no
Pass control required: no
Power-up state: STAR

Example Statements:

```
OUTPUT 711; "freq:ref star"
OUTPUT 711; "FREQUENCY;REFERENCE CENTER"
OUTPUT 711; "Freq:Ref?"
```

Command Syntax: FREQuency:REFerence <sp> {CENTer|STARt}

Query Syntax: FREQuency:REFerence?

Returned Format: {CENT|STAR} <LF> <^END>

Description:

When you change the analyzer's frequency span or time record length, the value of either FREQ:CENT or FREQ:STAR must be adjusted. This command lets you specify which of the two values should be held constant when such a change occurs.

If FREQ:REF is CENT, FREQ:CENT is held constant and FREQ:STAR is adjusted to make the following formula true:

$$\text{FREQ:STAR} = \text{FREQ:CENT} - (\text{FREQ:SPAN}/2)$$

If FREQ:REF is STAR, FREQ:STAR is held constant and FREQ:CENT is adjusted to make the following formula true:

$$\text{FREQ:CENT} = \text{FREQ:STAR} + (\text{FREQ:SPAN}/2)$$

FREQ:SPAN[?] command/query

Overlapped: no
Delayed result: yes
Pass control required: no
Power-up state: 102400

Example Statements:

```
OUTPUT 711;"Freq:Span 100"
OUTPUT 711;"Frequency:Span 10kHz"
OUTPUT 711;"freq:span?"
```

Command Syntax: FREQuency:SPAN<sp><value>[<unit>]

<value>::=any x, where x=max_span/2n

max_span::=102,400 for one-channel measurements,
51,200 for two-channel measurements

n::=an integer from 0 through 19

<unit>::=~HZ~ |KHZ

Query Syntax: FREQuency:SPAN?

Returned Format: <value><LF><^END>

<value>::=a decimal number (NRf format)

Description:

This command specifies the width of the band of frequencies you want to analyze. The value of FREQ:SPAN is used together with either FREQ:CENT or FREQ:STAR to completely define the band.

When you send this command, two other values are adjusted. SWE:TIME is adjusted so the following formula is true:

$$SWE:TIME = 400/FREQ:SPAN$$

The other value that is adjusted is either FREQ:CENT or FREQ:STAR, depending on which of the two values was last set. The value last set remains fixed while the other is adjusted to make the following formula true:

$$FREQ:SPAN = (FREQ:CENT - FREQ:STAR) \times 2$$

You can either use numbers or one of three nonnumeric parameters to set the value of FREQ:SPAN. The nonnumeric parameters are:

- UP – increases FREQ:SPAN span to the next largest allowable value
- DOWN – decreases FREQ:SPAN span to the next smallest allowable value
- (MARK[:A|:B]:VAL) – sets FREQ:SPAN to the closest allowable span that satisfies the following formula:

$$\text{FREQ:SPAN} \geq (\text{MARK:VAL})$$

- (MARK:VAL) will equal one of two values, depending on the marker mode selected. When MARK:X:MODE is NORM, (MARK:VAL) equals the value of the main marker. When MARK:X:MODE is DELT, (MARK:VAL) equals the absolute value of the difference between the marker reference value and the main marker value.

The query returns the width of the band of frequencies currently being analyzed. The value is returned in Hz.

FREQ:SPAN:FULL command

Overlapped: no
Delayed result: yes
Pass control required: no
Power-up state: not applicable

Example Statements:

```
OUTPUT 711;"FREQ:SPAN:FULL"
OUTPUT 711;"Frequency:Span:Full"
```

Command Syntax: FREQuency:SPAN:FULL

Description:

This command sets the start frequency (FREQ:STAR) to 0 Hz and the frequency span (FREQ:SPAN) to the largest allowable value. For one-channel measurements, the largest allowable span is 102.4 kHz. For two-channel measurements, the largest allowable span is 51.2 kHz.

FREQ:STARt[?] command/query

Overlapped: no
Delayed result: yes
Pass control required: no
Power-up state: 0

Example Statements:

```
OUTPUT 711;"freq:star 10"
OUTPUT 711;"FREQUENCY:START 50KHZ"
OUTPUT 711;"FREQ:STAR?"
```

Command Syntax: FREQuency:STARt<sp><value>[<unit>]

<value>::=any x, where 0 ≤ x ≤ max_disp_freq - (min_span/2)
Send numbers in the NRf format.
max_disp_freq::=115,000 Hz for one-channel measurements,
57,500 Hz for two-channel measurements
min_span::=0.1953 Hz for one-channel measurements
0.09766 Hz for two-channel measurements
<unit>::=~HZ~ |KHZ

Query Syntax: FREQuency:STARt?

Returned Format: <value><LF><^END>

<value>::=a decimal number (NRf format)

Description:

This command specifies the start of the band of frequencies you want to analyze. The values of FREQ:STAR and FREQ:SPAN completely define the band.

The value you send with this command also determines whether the analyzer is in the baseband or zoom mode measurement mode. When FREQ:STAR is 0 Hz, the analyzer is in baseband mode and time-domain data is real. When FREQ:STAR is anything other than 0 Hz, the analyzer is in zoom mode and time-domain data is complex.

When you send this command, the value of FREQ:CENT is automatically adjusted (if necessary) so that the following formula is true:

FREQ:CENT=FREQ:STAR+(FREQ:SPAN/2)

The same formula continues to adjust the value of FREQ:CENT each time FREQ:STAR or FREQ:SPAN is changed. This remains true until you explicitly set the value of FREQ:CENT or until you send FREQ:REF CENT, at which point FREQ:STAR becomes the value that is automatically adjusted.

You can either use numbers or one of three nonnumeric parameters to set the value of FREQ:STAR. The nonnumeric parameters are:

- UP – increases the current value of FREQ:STAR by the amount specified in FREQ:CENT:STEP
- DOWN – decreases the current value of FREQ:STAR by the amount specified in FREQ:CENT:STEP
- (MARK[:A|:B]:VAL) – sets FREQ:STAR to the frequency of the main marker, even when the marker reference is enabled

The query returns the current start of the band of frequencies being analyzed. The value is returned in Hz.

GPIB subsystem

Description:

Special fields on the analyzer's screen allow front-panel operators to monitor certain HP-IB functions. Commands in this subsystem are used to enable and disable display of these fields.

GPIB:LEDS[?] command/query

Overlapped: no
Delayed result: no
Pass control required: no
Power-up state: 0

Example Statements:

```
OUTPUT 711;"Gpib:Leds 0"
OUTPUT 711;"gpib:leds on"
OUTPUT 711;"GPIB:LEDS?"
```

Command Syntax: GPIB:LEDS<sp>{OFF|ON|0|1}

Query Syntax: GPIB:LEDS?

Returned Format: {0|1}<LF><^END>

Description:

Use this command to enable the display of the four HP-IB status indicators. When enabled, the indicators will appear in the upper-right corner of the analyzer's screen.

The indicators are Rmt, Tlk, Ltn, and Srq. Rmt brightens when the analyzer is under the control of an external controller. Tlk brightens when the analyzer is addressed to talk. Ltn brightens when the analyzer is addressed to listen. Srq brightens when the analyzer has issued a service request.

The query returns 0 if the status indicators are off, 1 if they are on.

GPIB:MNEMonic[?] command/query

Overlapped: no
Delayed result: no
Pass control required: no
Power-up state: OFF

Example Statements:

```
OUTPUT 711;"GPIB:MNEM ECHO"
OUTPUT 711;"GPIB:MNEMONIC SCROLL"
OUTPUT 711;"Gpib:Mnem?"
```

Command Syntax: GPIB:MNEMonic<sp>{ECHO|OFF|SCRoll}

Query Syntax: GPIB:MNEMonic?

Returned Format: {ECHO|OFF|SCR}<LF><^END>

Description:

A field in the upper-left portion of the analyzer's screen can be enabled to provide information on HP-IB commands. Use this command to enable the information field and to specify the type of HP-IB information you want displayed. The options are:

- OFF – This disables the information field.
- ECHO – This enables the information field and specifies that HP-IB programming mnemonics should be echoed to the field in response to front-panel key presses or to bus commands. ECHO is used most often to determine which HP-IB programming mnemonic is equivalent to a particular front-panel key sequence.
- SCR – This enables the information field and specifies that characters being sent to the analyzer over the bus should be scrolled into the field. As new characters are received, they are added to the right of the field. Old characters are scrolled off the screen to the left. When the analyzer's command parser recognizes an error, an all-white character is placed in the field. The all-white character is placed just after the character on which the error was recognized.

NOTE HP-IB transfers are much slower when SCR is selected. It should only be used when you are debugging programs or checking the integrity of bus transfers.

The query response tells you which option is currently selected.

INITialize subsystem

Description:

The single command in this subsystem is used to start, pause, and continue a measurement.

INIT:STATe[?] command/query

Overlapped: yes
Delayed result: no
Pass control required: no
Power-up state: RUN

Example Statements:

```
OUTPUT 711;"Init:Stat Run"
OUTPUT 711;"Initialize:State Start"
OUTPUT 711;"init:stat?"
```

Command Syntax: INITialize:STATe<sp>{PAUSe|RUN|STARt}

Query Syntax: INITialize:STATe?

Returned Format: {PAUS|RUN}<LF><^END>

Description:

This command is used to start, pause or continue a measurement.

NOTE INIT:STAT STAR and INIT:STAT RUN are considered to be pending overlapped commands whenever bit 7 of the Device Status condition register is set to 1. See Chapter 5 for a description of that bit.

INIT:STAT STAR starts a new measurement and ensures that changes made with delayed result commands are reflected in the measurement results. The new measurement is started immediately whether the current measurement is running, paused, or completed. All data from the previous measurement is discarded when the new measurement is started.

INIT:STAT PAUS pauses the current measurement. If the measurement is averaged, the current average is completed before the measurement is paused.

INIT:STAT RUN continues a paused measurement. It also allows you to add more data to the running average of a completed measurement. For example, if the analyzer has completed a 10-average measurement and you send INIT:STAT RUN, 10 more records are averaged in with the old data, bringing the total number of averages to 20.

If you send AVER:INIT followed by INIT:STAT RUN, the result is the same as if you send INIT:STAT STAR.

The query indicates whether the measurement is currently paused or running.

INPut subsystem

Description:

Commands in this subsystem are used to configure the inputs for channel 1 and channel 2.

Because there are two channels, you need to specify the channel you want to configure when you send a command. To specify the channel, append 1 or 2 to the word INPUT or to its short form INP. When you don't explicitly specify one of the channels in this manner, the analyzer configures channel 1.

NOTE The HP 35660A has two input channels (1 and 2) and two displays (A and B). However, neither of the two channels is linked to a particular display. You can display channel-1 data in either display A or display B. The same is true for channel-2 data.

INP:COUPling[?] command/query

Overlapped: no
Delayed result: yes
Pass control required: no
Power-up state: DC

Example Statements:

```
OUTPUT 711;"INP1:COUP AC"
OUTPUT 711;"Input2:Coupling dc"
OUTPUT 711;"Inp1:Coup?"
```

Command Syntax: INPut[~1~|2]:COUPling<sp>{AC|DC}

Query Syntax: INPut[~1~|2]:COUPling?

Returned Format: {AC|DC}<LF><^END>

Description:

This command selects AC or DC coupling for the specified channel.

The query response tells you whether AC or DC coupling is currently selected for the specified channel.

INP:IMPedance[?] command/query

Overlapped: no
Delayed result: yes
Pass control required: no
Power-up state: 50

Example Statements:
```
OUTPUT 711;"INP1:IMP 50"
OUTPUT 711;"INPUT2:IMPEDANCE .333"
OUTPUT 711;"Inp2:Imp?"
```

Command Syntax: INPut[~1~|2]:IMPedance<sp><ohms>[<unit>]

<ohms>::=any x, where 10E-3 ≤ x ≤ 10E+6 (when units are ohms)

Send values in NRf format.

<unit>::=~OHM~|KOHM|MOHM

Query Syntax: INPut[~1~|2]:IMPedance?

Returned Format: <value><LF><^END>

<value>::=a decimal number (NRf format)

Description:

Use this command to enter the impedance to be used for dBm calculations. The value you select is used for both channels regardless of the channel you specify.

You can either use numbers or one of two nonnumeric parameters to set the value of INP:IMP. The nonnumeric parameters are:

- UP – changes the current value of INP:IMP to the next largest value defined by the analyzer (The analyzer defines input impedance values that are 1, 2, and 5 times the powers of 10 that fall within the acceptable range.)
- DOWN – changes the current value of INP:IMP to the next smallest value defined by the analyzer

The query returns the impedance value currently being used for dBm calculations. The value is returned in ohms.

INP:LOW[?] command/query

Overlapped: no
Delayed result: yes
Pass control required: no
Power-up state: GRO

Example Statements:

```
OUTPUT 711;"inp1:low gro"
OUTPUT 711;"INPUT2:LOW FLO"
OUTPUT 711;"INP2:LOW?"
```

Command Syntax: INPut[~1~|2]:LOW<sp>{FLOat|GROund}

Query Syntax: INPut[~1~|2]:LOW?

Returned Format: {FLO|GRO}<LF><^END>

Description:

Use this command to float or ground the specified channel's input shield.

A floated input shield is connected to ground through 1 Megohm. A grounded shield is connected to ground through 55 ohms.

The query returns FLO if the specified input's shield is floated, GRO if it is grounded.

INP:RANGe[?] command/query

Overlapped: no
Delayed result: yes
Pass control required: no
Power-up state: variable

Example Statements:

```
OUTPUT 711;"Inp2:Rang 10"
OUTPUT 711;"input1:range 6.2dbm"
OUTPUT 711;"INP1:RANG?"
```

Command Syntax: INPut[~1~|2]:RANGe<sp><value>[<unit>]

<value>::=a decimal number (NRf format)

<unit>::=~V~|VRMS|DBM|DBVPK|DBVRMS (when INP:UNIT is VOLT)

~EU~|EURMS|DBEUPK|DBEURMS (when INP:UNIT is EU)

Query Syntax: INPut[~1~|2]:RANGe?

Returned Format: <value><LF><^END>

Description:

Use this command to enter a range for the specified channel.

Valid input ranges are from 27 through −51 dBVrms in 2 dB steps. If you send a value with this command, it is rounded up to the next highest range. If you do not specify units when you send a new value, the default unit is assumed.

You can either use a number or one of the three nonnumeric parameters to set the value of INP:RANG. The nonnumeric parameters are:

- UP – changes INP:RANG to the next highest valid range
- DOWN – changes INP:RANG to the next lowest valid range
- (MARK[:A|:B]:VAL) – sets INP:RANG to the nearest value that is greater than or equal to the amplitude of the main marker, even when the marker reference is enabled

---

NOTE You can specify a new value for the channel-2 range while you are in the one-channel measurement mode. However, the value is not used to set the channel-2 range until you enter the two-channel mode.

The query response tells you which range is currently selected for the specified channel. The value is returned in the units last used to set the range.

INP:RANG:AUTO[?] command/query

Overlapped: yes
Delayed result: no
Pass control required: no
Power-up state: 1

Example Statements:

```
OUTPUT 711;"Inp1:Rang:Auto 1"
OUTPUT 711;"Input2:Range:Auto On"
OUTPUT 711 "INP:RANG:AUTO?"
```

Command Syntax: INPut[~1~|2]:RANGe:AUTO:<sp> {OFF|ON|0|1}

Query Syntax: INPut {~|~|2}:RANGe:AUTO?

Returned Format {0|1} <LF> <^END>

Description:

This command enables and disables the autoranging routine for the specified channel.

NOTE The analyzer never autoranges while an averaged measurement is in progress.

The autorange routine starts by selecting the lowest input range. It then steps the input up through successive ranges until the input is no longer overloaded. The routine continues to adjust the range upward in response to increased signal amplitude.

The autorange routine never adjusts the range downward in response to decreased signal amplitude. You must restart the autorange routine if you think the range is too large for the current input signal. Sending INP:RANG:AUTO ON restarts the autorange routine, even if autoranging is already turned on.

Autoranging is disabled in one of the following ways:

- Sending INP:RANG:AUTO OFF (or O)
- Specifying a range with the INP:RANG command

If you use INP:RANGE:AUTO OFF to disable autoranging, the range is fixed at the last value selected by the autoranging routine.

The query returns 0 if autoranging is disabled, 1 if it is enabled.

INP:UNITs[?] command/query

Overlapped: no
Delayed result: yes
Pass control required: no
Power-up state: VOLT

Example Statements:
```
OUTPUT 711;"INP1:UNIT EU"
OUTPUT 711;"Input2:Units Volt"
OUTPUT 711;"Inp1:Unit?"
```

Command Syntax: INPut[~1~|2]:UNITs<sp>{EU|VOLT}

Query Syntax: INPut[~1~|2]:UNITs?

Returned Format: {EU|VOLT}<LF><^END>

Description:

The analyzer allows you to specify input ranges and vertical scaling parameters in either volts or engineering units. Use this command to select the kind of units you want to use.

Use INP:UNIT:EU:MULT to specify a scaling factor (V/EU) for relating EU values to volts. Use INP:UNIT:EU:NAME to enter the name of the engineering unit you are using. The name is used to label the vertical axis of trace displays.

The query response indicates whether input range and vertical scaling parameters are currently being interpreted in volts or engineering units.

INP:UNIT:EU selector

Description:

This command only selects the INP:UNIT:EU subsystem. Sending INP:UNIT:EU alone does nothing.

INP:UNIT:EU:MULTiplier[?] command/query

Overlapped: no
Delayed result: yes
Pass control required: no
Power-up state: 1

Example Statements:

```
OUTPUT 711;"inp1:unit:eu:mult 1"
OUTPUT 711;"INPUT2:UNITS:EU:MULTIPLIER 1.5"
OUTPUT 711;"INP2:UNIT:EU:MULT?"
```

Command Syntax: INPut[~1~|2]:UNITs:EU:MULTiplier<sp><volts_per_eu>

<volts_per_eu>::=a decimal number (NRf format)

Query Syntax: INPut[~1~|2]:UNITs:EU:MULTiplier?

Returned Format: <volts_per_eu><LF><^END>

Description:

This command allows you to specify an engineering unit (EU) scaling factor. The factor you enter represents the number of volts per engineering unit (V/EU).

The EU scaling factor is only used when INP:UNIT is set to EU. When used, the factor relates engineering unit values to volts measured at the specified input channel.

The query returns the scaling factor currently being used for the specified channel.

INP:UNIT:EU:NAME[?] command/query

Overlapped: no
Delayed result: yes
Pass control required: no
Power-up state: "EU"

Example Statements:

```
OUTPUT 711;"INP1:UNIT:EU:NAME ""KNOT"""
OUTPUT 711;"INPUT1:UNITS:EU:NAME 'g'"
OUTPUT 711;"Inp1:Unit:Eu:Name?"
```

Command Syntax: INPut[~1~|2]:UNIT:EU:NAME<sp>{'|"}<name>{'|"}

<name>::=1 to 8 ASCII characters

Query Syntax: INPut[~1~|2]:UNIT:EU:NAME?

Returned Format: "<name>"<LF><^END>

Description:

This command allows you to enter a name for the specified channel's engineering units. The name is used to label the vertical axis of trace displays when INP:UNIT is set to EU.

The following unit labels are reserved and cannot be entered as an engineering unit label:

VRMS, V^2, VRMS^2, V^2/HZ, V/RTHZ, VRMS^2/HZ, VRMS/RTHZ, DB, DBM/HZ, DBVPK, DBVRMS, DBM, DBVPK/RTHZ, DBVRMS/RTHZ.

The query returns the engineering unit name last entered for the specified channel. The name is returned as an ASCII character string.

LIMit subsystem

Description:

This subsystem provides access to limit tables over the HP-IB. It does not provide access to the result of limit tests. For limit test results, see the DISPlay subsystem.

LIM:TABLe[?] command/query

Description:

As a result, LIM:TABL is functionally equivalent to LIM:TABL:DATA. See the latter command for more details.

LIM:TABL[:DATA][?] command/query

Overlapped: no
Delayed result: no
Pass control required: no
Power-up state: 0 (x_start)
0 (x_stop)
0 (y_start)
0 (y_stop)
2 (y_flag)

Example Statements:

```
OUTPUT 711;"Lim1:Tabl 2000,3000,.3,.5,2"
OUTPUT 711;"limit7:table:data 50,55,.1,.45,1"
OUTPUT 711;"LIM:TABL:DATA?"
```

Command Syntax: LIMit[<table#>]:TABLe[:DATA]<sp><block_data>

<table#>::=1 through 8 (NRf format)

<block_data> takes one of two forms depending on whether you are sending ASCII-encoded or binary-encoded data. When data is ASCII-encoded, (LIM:TABL:HEAD:AFOR ASC):

<block_data>::={,}...<LF><^END>

::=<x_start>,<x_stop>,<y_start>,<y_stop>,<y_flag>

Send these values for as many segment as you plan to define.

Send all values in the NRf format and separate them with commas.

When data is binary-encoded, (LIM:TABL:HEAD:AFOR FP32 or LIM:TABL:HEAD:AFOR FP64):

<block_data>::=#<byte><length_bytes>{}...

<byte>::=one ASCII-encoded byte that specifies the number of length bytes to follow

<length_bytes>::=ASCII-encoded bytes that specify the number of data bytes to follow

::=<x_start><x_stop><y_start><y_stop><y_flag>

Send all values as either 32-bit or 64-bit binary floating point numbers (depending on the setting of LIM:TABL:HEAD:AFOR).

Query Syntax: LIMit<table#>:TABLe[:DATA]?

Returned Format: <block_data>

NOTE To determine the number of segments that will be returned in <block_data>, use the LIM:TABL:HEAD:POIN query.

Description:

Use this command to define the specified limit table. The table can then be coupled to one of the displays, and data in the display can be tested against the defined limits.

Each in a limit table defines a line segment. Each line segment serves as an upper or lower boundary for acceptable y-axis values over a certain range of x-axis values. You send as many segments as are required to define the limits.

Each segment consists of an x_start, x_stop, y_start, y_stop, and y_flag value. The x_start and y_start values define the beginning point of a line segment on the coordinate plane. The x_stop and y_stop values define the ending point of the segment on the coordinate plane. The y_flag value indicates whether the segment defines the upper or lower boundary of acceptable y-axis values, 2 being upper, 1 being lower.

You define limits assuming that the data to be tested will be displayed using a particular x-axis unit and y-axis unit. However, limit table values are actually unitless. A y_start value of 4, for example, will be 4 dBVrms if data is displayed using dBVrms on the y-axis. It will be 4 degrees if data is displayed using degrees on the y-axis. So before you test data against a limit table, you must be sure that display units are the same as the units you assumed when defining the table.

The query returns all segments currently defined for the specified limit table. The format of the returned data depends on the setting of LIM:TABL:HEAD:AFOR.

LIM:TABL:HEADer selector

Description:

This command only selects the LIM:TABL:HEAD subsystem. Sending LIM:TABL:HEAD alone does nothing.

LIM:TABL:HEAD:AFORmat[?] command/query

Overlapped: no
Delayed result: no
Pass control required: no
Power-up state: ASC

Example Statements:
```
OUTPUT 711;"Lim2:Tabl:Head:Afor Asc"
OUTPUT 711;"Limit8:Table:Header:Aformat Fp64"
OUTPUT 711;"lim:tabl:head:afor?"
```

Command Syntax: LIMit[<table #>]:TABLe:HEADer:AFORmat<sp>{ASCii|FP32|FP64}

<table#>::=1 through 8 (NRf format)

Query Syntax: LIMit[<table #>]:TABLe:HEADer:AFORmat?

Returned Format: {ASC|FP32|FP64}<LF><^END>

Description:

Limit tables can either be ASCII-encoded or binary-encoded when they are transferred between the analyzer and an HP-IB controller. This command lets you specify how the display data should be encoded.

When ASC is selected, data is sent as a series of values separated by commas. The values are ASCII-encoded and are formatted as NRf decimal numbers.

FP32 and FP64 both specify binary encoding. When FP32 is selected, data is sent as a series of values within a definite length block. The values are encoded as 32-bit binary floating point numbers. When FP64 is selected, data is also sent as a series of values within a definite length block. However, the values are encoded as 64-bit binary floating point numbers.

For more information on data encoding and data transfer formats, see Chapter 4, "Transferring Data."

The query returns ASC, FP32, or FP64 depending on the option currently specified.

LIM:TABL:HEAD:POINts? query

Overlapped: no
Delayed result: no
Pass control required: no
Power-up state: 1

Example Statement: `OUTPUT 711;"LIM:TABL:HEAD:POIN?"`

Query Syntax: LIMit[<table #>]:TABLe:HEADer:POINts?

<table#>::=1 through 8 (NRf format)

Returned Format: <value><LF><^END>

<value>::=an integer (NR1 format)

Description:

A limit table returns segments in response to the LIM:TABL:DATA query. Use this query (LIM:TABL:HEAD:POIN) to determine how many segments will be returned from the specified limit table.

MARKer subsystem

Description:

Commands in this subsystem are used to access the analyzer's marker functions and marker data.

With a few exceptions, marker commands must be directed to one of the two displays: A or B. To specify a display, insert one of the following between the MARKER or MARK and the rest of the command:

- :A – as in MARK:A:BAND:CENT 51200
- :B – as in MARKER:B:HARMONIC:POWER?
- 1 – as in MARKER1:X:AUTO OFF
- 2 – as in MARK2:X:POIN?

Using :A or 1 directs the command to display A. Using :B or 2 directs the command to display B. If you don't explicitly specify one of the displays, the command is directed to display A.

NOTE The display to which you direct a command becomes the active display.

Two marker functions that are not found in this subsystem are limit tables and data tables. Limit table commands are found in the DISPlay and LIMit subsystems. Data table commands are found in the TRACe subsystem.

MARK:BAND selector

Description:

This command only selects the MARK:BAND subsystem. Sending MARK:BAND alone does nothing.

MARK:BAND:CENTer[?] command/query

Overlapped: no
Delayed result: no
Pass control required: no
Power-up state: 51200

Example Statements:

```
OUTPUT 711;"mark1:band:cent 100hz"
OUTPUT 711;"MARKER:B:BAND:CENTER 51.2KHZ"
OUTPUT 711;"MARK:BAND:CENT?"
```

Command Syntax: MARKer[<spec>]:BAND:CENTer<sp><value>[<unit>]

<spec>::=~:A~|:B|1|2

<value>::=any x, where 0 ≤ x ≤ 115 kHz for a 1-channel measurement.

any x, where 0 ≤ x ≤ 57.5 kHz for a 2-channel measurement.

Send all numbers in NRf format.

<unit>::=~HZ~|KHZ

Query Syntax: MARKer[<spec>]:BAND:CENTer?

Returned Format: <value><LF><^END>

Description:

This command defines the center of a band frequencies in which power is to be calculated.

You can either use numbers or one of three nonnumeric parameters to set the value of MARK:BAND:CENT. The nonnumeric parameters are:

- UP – steps MARK:BAND:CENT to the value of the next largest point on the x-axis
- DOWN – steps MARK:BAND:CENT to the value of the next smallest point on the x-axis
- (MARK[:A|:B]:VAL) – sets MARK:BAND:CENT to the frequency of the main marker, even when the marker reference is enabled

---

NOTE When you change the value of MARK:BAND:CENT, the values of MARK:BAND:STAR and MARK:BAND:STOP are changed by the same amount. This shifts the entire band up or down in frequency.

The query returns the center frequency of the band in Hz. The value is returned even if the band markers are not enabled.

MARK:BAND:POWer? query

Overlapped: no
Delayed result: no
Pass control required: no
Power-up state: 0

Example Statement: `OUTPUT 711;"MARK:A:BAND:POW?"`

Query Syntax: MARKer[<spec>]:BAND:POWer?

<spec>::=~:A~|:B|1|2

Returned Format: <value><LF><^END>

<value>::=a decimal number (NRf format)

Description:

Power can be calculated for the band of frequencies defined by the band markers. This query returns the results of the power calculation.

Power is calculated as an rms summation of the power at each frequency within the band. In order for this query to return the result of a power calculation, MARK:BAND:STAT and MARK:BAND:POW:STAT must be ON. The result is converted to the current vertical unit (DISP:Y:SCAL:UNIT) after power is calculated.

The query only returns a value. Units are returned with the DISP:Y:SCAL:UNIT query.

MARK:BAND:POW:STATe[?] command/query

Overlapped: no
Delayed result: no
Pass control required: no
Power-up state: 1

Example Statements:

```
OUTPUT 711;"Mark:Band:Pow:Stat 0"
OUTPUT 711;"marker:band:power:state on"
OUTPUT 711;"MARK:BAND:POW:STAT?"
```

Command Syntax: MARKer[<spec>]:BAND:POWer:STATe<sp>{OFF|ON|0|1}

<spec>::=~:A~|:B|1|2

Query Syntax: MARKer[<spec>]:BAND:POWer:STATe?

Returned Format: {0|1}<LF><^END>

Description:

Use this command to enable or disable the band power calculation. When enabled, power is calculated for the band of frequencies defined by the band markers and the results of the calculation are displayed on the analyzer's screen. See MARK:BAND:POW for more information.

The query returns 0 if the band power calculation is disabled for the specified display, 1 if it is enabled.

MARK:BAND:STARt[?] command/query

Overlapped: no
Delayed result: no
Pass control required: no
Power-up state: 46080

Example Statements:

```
OUTPUT 711;"Mark2:Band:Star 1kHz"
OUTPUT 711;"Marker:Band:Start 57hz"
OUTPUT 711;"mark1:band:star?"
```

Command Syntax: MARKer[<spec>]:BAND:STARt<sp><value>[<unit>]

<spec>::=~:A~|:B|1|2

<value>::=any x, where 0 ≤ x ≤ 115 kHz for a one-channel measurement.

any x, where 0 ≤ x ≤ 57.5 kHz for a two-channel measurement.

Values must be decimal numbers in NRf format.

<unit>::=~HZ~|KHZ

Query Syntax: MARKer[<spec>]:BAND:STARt?

Returned Format: <value><LF><^END>

Description:

This command defines the lowest frequency of the band in which power is to be calculated.

You can either use numbers or one of three nonnumeric parameters to set the value of MARK:BAND:STAR. The nonnumeric parameters are:

- UP – steps MARK:BAND:STAR to the value of the next largest point on the x-axis
- DOWN – steps MARK:BAND:STAR to the value of the next smallest point on the x-axis
- (MARK[:A|:B]:VAL) – sets MARK:BAND:STAR to the frequency of the main marker, even when the marker reference is enabled

The query returns the lowest frequency of the band in Hz. The value is returned even if the band markers are not enabled.

MARK:BAND:STATe[?] command/query

Overlapped: no
Delayed result: no
Pass control required: no
Power-up state: 0

Example Statements:
```
OUTPUT 711;"MARK2:BAND:STAT OFF"
OUTPUT 711;"Marker:B:Band:State 1"
OUTPUT 711;"Mark:B:Band:Stat?"
```

Command Syntax: MARKer[<spec>]:BAND:STATe<sp>{OFF|ON|0|1}

<spec>::=~:A~|:B|1|2

Query Syntax: MARKer[<spec>]:BAND:STATe?

Returned Format: {0|1}<LF><^END>

Description:

This command enables and disables the band markers. Band markers must be enabled (MARK:BAND:STAT ON) before the results of a band power calculation can be returned. See MARK:BAND:POW for more information.

At any given time, only one of the following markers can be active in the specified display:

- Band (MARK:BAND:STATE)
- Harmonic (MARK:HARM:STATE)
- Sideband (MARK:SID:STATE)

If you enable the band markers when one of the other two is already enabled, that other marker is automatically disabled.

The query returns 0 if the specified display's band markers are disabled, 1 if they are enabled.

MARK:BAND:STOP[?] command/query

Overlapped: no
Delayed result: no
Pass control required: no
Power-up state: 56320

Example Statements:
```
OUTPUT 711;"mark:a:band:stop 10000 Hz"
OUTPUT 711;"MARKER1:BAND:STOP 75KHZ"
OUTPUT 711;"MARK1:BAND:STOP?"
```

Command Syntax: MARKer[<spec>]:BAND:STOP<sp><value>[<unit>]

<spec>::=~:A~|:B|1|2

<value>::=any x, where 0 ≤ x ≤ 115 kHz for a one-channel measurement.

any x, where 0 ≤ x ≤ 57.5 kHz for a two-channel measurement.

Values must be decimal numbers in NRf format.

<unit>::=~HZ~|KHZ

Query Syntax: MARKer[<spec>]:BAND:STOP?

Returned Format: <value><LF><^END>

Description:

This command defines the highest frequency of the band in which power is to be calculated.

You can either use numbers or one of three nonnumeric parameters to set the value of MARK:BAND:STOP. The nonnumeric parameters are:

- UP – steps MARK:BAND:STOP to the value of the next largest point on the x-axis
- DOWN – steps MARK:BAND:STOP to the value of the next smallest point on the x-axis
- (MARK[:A|:B]:VAL) – sets MARK:BAND:STOP to the frequency of the main marker, even when the marker reference is enabled

The query returns the highest frequency of the band in Hz. The value is returned even if the band markers are not enabled.

MARK:DTABle command

Description:

MARK:DTAB is functionally equivalent to MARK:DTAB:DATA. See the latter command for more details.

MARK:DTAB[:DATA][?] command/query

Overlapped: no
Delayed result: no
Pass control required: no
Power-up state: 0,0

Example Statements:

```
OUTPUT 711;"MARKER:DTABLE:DATA 12800,25600"
OUTPUT 711;"MARK2:dtab:data?"
```

Command Syntax: MARKer[<spec>]:DTABle[:DATA]<sp><block_data>

<spec>::=~:A~|:B|1|2

<block_data> takes one of two forms, depending on whether the data is ASCII-encoded or binary-encoded. When data is ASCII-encoded (MARK:DTAB:HEAD:AFOR ASC):

<block_data>::={<x_value>,}...<x_value n><LF><^END>

<x_value>::=x-axis values for the 1st through nth points

Send values using the NRf format. The values you send are understood to be in the current display units.

When data transfers are set to binary (MARK:DTAB:HEAD:AFOR FP32 or MARK:DTAB:HEAD:AFOR FP64):

<block_data>::=#<byte><length_bytes>{<x_value>}...

<byte>::=one ASCII-encoded byte that specifies the number of length bytes to follow

<length_bytes>::=ASCII-encoded bytes that specify the number of data bytes to follow

<x_value> has the same definition specified for ASCII-encoded data. However, you should send it as either a 32-bit or a 64-bit binary floating point number, depending on how you have set MARK:DTAB:HEAD:AFOR.

Query Syntax: MARKer[<spec>]:DTABle[:DATA]?

Returned Format: <block_data>

<block_data> takes one of two forms, depending on whether the data is ASCII-encoded or binary-encoded. When data is ASCII-encoded (MARK:DTAB:HEAD:AFOR ASC):

<block_data>::={<point>,}...<point n><LF><^END>

<point>::=<x_value>,<y_value>

<x_value>::=x-axis values for the 1st through nth points (n is returned with the MARK:DTAB:HEAD:POIN query)

<y_value>::=y-axis values for the 1st through nth points

Values are returned in the NRf format. Units for the values are the current display units.

When data transfers are set to binary (MARK:DTAB:HEAD:AFOR FP32 or MARK:DTAB:HEAD:AFOR FP64):

<block_data>::=#<byte><length_bytes>{<point>}...

<byte>::=one ASCII-encoded byte that specifies the number of length bytes to follow

<length_bytes>::=ASCII-encoded bytes that specify the number of data bytes to follow

<point>::=<x_value><y_value>

<x_value> and <y_value> have the same definition specified for ASCII-encoded data. However, they are returned as either 32-bit or 64-bit binary floating point numbers, depending on the setting of MARK:DTAB:HEAD:AFOR.

Description:

Use this command to define a data table for the specified trace.

A data table is defined by sending a series of x-axis values. The values are assumed to be in the current x-axis units (TRAC:HEAD:XUN). When data table calculation is turned on (MARK:DTAB:STAT ON), the y-axis value of the trace is calculated for each x-axis value that you sent.

The query returns a series of points. Each point consists of an x-axis value followed by the y-axis value of the trace at that point. Use TRAC:HEAD:XUN to determine the units for the x-axis values. Use TRAC:HEAD:YUN to determine the units for the y-axis values. Y-axis values will all be 0 if MARK:DTAB:STAT is OFF.

MARK:DTAB:HEADer — selector

Description:

This command only selects the MARK:DTAB:HEAD subsystem. Sending MARK:DTAB:HEAD alone does nothing.

MARK:DTAB:HEAD:AFORmat[?] — command/query

Overlapped: no
Delayed result: no
Pass control required: no
Power-up state: ASC

Example Statements:

```
OUTPUT 711;"MARK2:DTAB:HEAD:AFOR FP64"
OUTPUT 711;"Marker1:Dtable:Header:Aformat ASCii"
OUTPUT 711;"MARK:B:DTAB:HEAD:AFOR?"
```

Command Syntax: MARKer[<spec>]:DTABle:HEADer:AFORmat<sp>{ASCii|FP32|FP64}

<spec>::=~:A~|:B|1|2

Query Syntax: MARKer[<spec>]:DTABle:HEADer:AFORmat?

Returned Format: {ASC|FP32|FP64}<LF><^END>

Description:

Data tables can either be ASCII-encoded or binary-encoded when they are transferred between the analyzer and an external controller. This command lets you specify how a data table should be encoded.

When ASC is selected, data is transferred as a series of values separated by commas. The values are ASCII-encoded and are formatted as NRf decimal numbers.

FP32 and FP64 both specify binary encoding. When FP32 is selected, data is sent as a series of values within a definite length block. The values are encoded as 32-bit binary floating point numbers. When FP64 is selected, data is also sent as a series of values within a definite length block. However, the values are encoded as 64-bit binary floating point numbers.

The query returns ASC, FP32, or FP64, depending on the option currently specified.

MARK:DTAB:HEAD:POINts? query

Overlapped: no
Delayed result: no
Pass control required: no
Power-up state: 1

Example Statement:
```
OUTPUT 711;marK:B:dtab:head:poin?"
```

Query Syntax: MARKer[<spec>]:DTABle:HEADer:POINts?

<spec>::=~:A~|:B|1|2

Returned Format: <value><LF><^END>

<value>::=an integer (NR1 format)

Description:

This query tells you how many data table points will be returned from the specified Trace in response to the MARK:DTAB:DATA query. Each point will contain both an x-axis value and a y-axis value.

MARK:DTAB:STATe[?] command/query

Overlapped: no
Delayed result: no
Pass control required: no
Power-up state:

Example Statements:
```
OUTPUT 711;"MARK:A:DTAB:STAT OFF"
OUTPUT 711;"MARKER:B:DTABLE:STATE 1"
OUTPUT 711;"Mark1:Dtab:Stat?"
```

Command Syntax: MARKer[<spec>]:DTABle:STATe<sp>{OFF|ON|0|1}

<spec>::=~:A~|:B|1|2

Query Syntax: TRACe[<spec>]:DTABle:STATe?

Returned Format: {0|1}<LF><^END>

Description:

This command enables data table calculations for the specified trace.

A data table is defined by one or more x-axis values. When data table calculations are enabled, the trace's y-axis value is calculated for each of the x-axis values. This calculation occurs each time the trace is updated.

The query returns 0 if data table calculations are disabled, 1 if they are enabled.

MARK:FUNCtion command

Overlapped: no
Delayed result: no
Pass control required: no
Power-up state: not applicable

Example Statement:
```
OUTPUT 711;"mark:b:func aoff"
```

Command Syntax: MARKer[<spec>]:FUNCtion AOFF

<spec>::=~:A~|:B|1|2

Description:

This command allows you to simultaneously turn off all special markers for the specified channel. The following are all set to 0:

- MARK:BAND:STAT
- MARK:HARM:STAT
- MARK:SID:STAT

MARK:HARMonic[?] command/query

Description:

MARK:HARM is functionally equivalent to MARK:HARM:FREQ. See the latter command for more details.

MARK:HARM:COUNt[?] command/query

Overlapped: no
Delayed result: no
Pass control required: no
Power-up state: 20

Example Statements:
```
OUTPUT 711;"MARK2:HARM:COUN 10"
OUTPUT 711;"MARKER:A:HARMONIC:COUNT 120"
OUTPUT 711;"Mark:B:Harm:Coun?"
```

Command Syntax: MARKer[<spec>]:HARMonic:COUNt<sp><value>

<spec>::=~:A~|:B|1|2

<value>::=any integer x, where $0 \leq x \leq 400$ (NRf format)

Query Syntax: MARKer[<spec>]:HARMonic:COUNt?

Returned Format: <value><LF><^END>

<value>::=an integer (NR1 format)

Description:

Use this command to specify the number of harmonic markers you want displayed.

You can either use numbers or one of two nonnumeric parameters to set the value of MARK:HARM:COUN. The nonnumeric parameters are:

- UP – increases the value of MARK:HARM:COUN by one
- DOWN – decreases the value of MARK:HARM:COUN by one

The query returns the number of harmonic markers currently specified for the display. The value is returned even if harmonic markers are not enabled.

MARK:HARM[:FREQuency][?] command/query

Overlapped: no
Delayed result: no
Pass control required: no
Power-up state: 10240

Example Statements:

```
OUTPUT 711;"Mark1:Harm 100"
OUTPUT 711;"marker:b:harmonic:frequency 25.7khz"
OUTPUT 711;"MARK2:HARM:FREQ?"
```

Command Syntax: MARKer[<spec>]:HARMonic[:FREQuency]<sp><value>[<unit>]

<spec>::=~:A~|:B|1|2

<value>::=any x, where 0 ≤ x ≤ 115 kHz for a one-channel measurement

any x, where 0 ≤ x ≤ 57.5 kHz for a two-channel measurement

Values must be decimal numbers in NRf format.

<unit>::=~HZ~|KHZ

Query Syntax: MARKer[<spec>]:HARMonic[:FREQuency]?

Returned Format: <value><LF><^END>

Description:

This command allows you to specify the fundamental frequency for harmonic markers and calculations.

You can either use numbers or one of three nonnumeric parameters to set the value of MARK:HARM:FREQ. The nonnumeric parameters are:

- UP – steps MARK:HARM:FREQ to the value of the next largest point on the x-axis
- DOWN – steps MARK:HARM:FREQ to the value of the next smallest point on the x-axis
- (MARK[:A|:B]:VAL) – sets MARK:HARM:FREQ to the frequency of the main marker, even when the marker reference is enabled

The query returns the fundamental frequency currently being used for harmonic markers and calculations. The value is returned in Hz. It is returned even if the harmonic markers are not enabled.

MARK:HARM[:FREQ]:DIVide command

Overlapped: no
Delayed result: no
Pass control required: no
Power-up state: not applicable

Example Statements:

```
OUTPUT 711;"Mark:Harm:Div 3"
OUTPUT 711;"MARK:B:HARM:FREQ:DIV 4"
```

Command Syntax: MARKer[<spec>]:HARMonic[:FREQuency]:DIVide<sp><value>

<spec>::=~:A~|:B|1|2

<value>::=any x, where 1 ≤ x ≤ 1E+100

Description:

This command allows you to divide the current value of MARK:HARM:FREQ by a specified amount. The result of this division becomes the new value of MARK:HARM:FREQ.

MARK:HARM:POWer? query

Overlapped: no
Delayed result: no
Pass control required: no
Power-up state: 0

Example Statement: `OUTPUT 711;"Mark:B:Harm:Pow?"`

Query Syntax: MARKer[<spec>]:HARMonic:POWer?

<spec>::=~:A~|:B|1|2

Returned Format: <value><LF><^END>

<value>::=a decimal number (NRf format)

Description:

Total harmonic power can be calculated for the current fundamental frequency. This query returns the results of the power calculation.

Total harmonic power is calculated as an rms summation of the power at all marked harmonics. The result is converted to the current vertical unit (DISP:Y:SCAL:UNIT) after power is calculated. In order for this query to return the result of a power calculation, MARK:HARM:STAT and MARK:HARM:POW:STAT must be ON.

The query only returns a value. Units are returned with the DISP:Y:SCAL:UNIT query.

MARK:HARM:POW:STATe[?] command/query

Overlapped: no
Delayed result: no
Pass control required: no
Power-up state: 0

Example Statements:

```
OUTPUT 711;"MARK:A:HARM:POW:STAT ON"
OUTPUT 711;"Marker:B:Harmonic:Power:State 0"
OUTPUT 711;"Mark2:Harm:Pow:Stat?"
```

Command Syntax: MARKer[<spec>]:HARM:POWer:STATe<sp>{OFF|ON|0|1}

<spec>::=~:A~|:B|1|2

Query Syntax: MARKer[<spec>]:HARM:POWer:STATe?

Returned Format: {0|1}<LF><^END>

Description:

Use this command to enable and disable the total harmonic power calculation. When enabled, power is calculated for the marked harmonics and the results of the calculation are displayed on the analyzer's screen. See MARK:HARM:POW for more information.

At any given time, only one of the two harmonic calculations can be enabled for the specified display. If you enable the harmonic power calculation when the total harmonic distortion calculation is already enabled, the latter is automatically disabled. (The total harmonic distortion calculation is enabled and disabled with the MARK:HARM:THD:STAT command.)

The query returns 0 if total harmonic power calculation is disabled for the specified display, 1 if it is enabled.

MARK:HARM:STATe[?] command/query

Overlapped: no
Delayed result: no
Pass control required: no
Power-up state: 0

Example Statements:

```
OUTPUT 711;"mark2:harm:stat on"
OUTPUT 711;"MARKER:B:HARMONIC:STATE 1"
OUTPUT 711;"MARK:A:HARM:STAT?"
```

Command Syntax: MARKer[<spec>]:HARMonic:STATe<sp>{OFF|ON|0|1}

<spec>::=~:A~|:B|1|2

Query Syntax: MARKer[<spec>]:HARMonic:STATe?

Returned Format: {0|1}<LF><^END>

Description:

This command enables and disables the harmonic markers. Harmonic markers must be enabled (MARK:HARM:STAT ON) before the results of a harmonic calculation can be returned. See MARK:HARM:POW and MARK:HARM:THD for more information.

At any given time, only one of the following markers can be active in the specified display:

- Band (MARK:BAND:STATE)
- Harmonic (MARK:HARM:STATE)
- Sideband (MARK:SID:STATE)

If you enable the harmonic markers when one of the other two is already enabled, that other marker is automatically disabled.

The query returns 0 if the specified display's harmonic markers are disabled, 1 if they are enabled.

MARK:HARM:THD? query

Overlapped: no
Delayed result: no
Pass control required: no
Power-up state: 0

Example Statement: `OUTPUT 711;"MARK1:HARM:THD?"`

Query Syntax: MARKer[<spec>]:HARMonic:THD?

<spec>::=~:A~|:B|1|2

Returned Format: <value><LF><^END>

<value>::=a decimal number (NRf format)

Description:

Total harmonic distortion can be calculated for the current fundamental frequency. This query returns the results of that calculation.

Total harmonic distortion (THD) expresses total harmonic power (THP) as a percentage of the power in the fundamental frequency. The formula is:

$$THD=(THP/fundamental\_power)\times 100$$

In order for this query to return the result of the THD calculation, MARK:HARM:STAT and MARK:HARM:THD:STAT must be ON.

MARK:HARM:THD:STATe[?] command/query

Overlapped: no
Delayed result: no
Pass control required: no
Power-up state: 1

Example Statements:

```
OUTPUT 711;"Mark:B:Harm:Thd:Stat Off"
OUTPUT 711;"marker1:harmonic:thd:state 1"
OUTPUT 711;"MARK2:HARM:THD:STAT?"
```

Command Syntax: MARKer[<spec>]:HARMonic:THD:STATe<sp>{OFF|ON|0|1}

<spec>::=~:A~|:B|1|2

Query Syntax: MARKer[<spec>]:HARMonic:THD:STATe?

Returned Format: {0|1}<LF><^END>

Description:

Use this command to enable and disable the total harmonic distortion (THD) calculation. When enabled, THD is calculated for the fundamental frequency and all marked harmonics. The results of the calculation are displayed on the analyzer's screen. See MARK:HARM:THD for more information.

At any given time, only one of the two harmonic calculations can be enabled for the specified display. If you enable the total harmonic distortion calculation when the harmonic power calculation is already enabled, the latter is automatically disabled. (The harmonic power calculation is enabled and disabled with the MARK:HARM:POW:STAT command.)

The query returns 0 if total harmonic distortion calculation is disabled for the specified display, 1 if it is enabled.

MARK:SIDeband[?] command/query

Description:

MARK:SID is functionally equivalent to MARK:SID:FREQ. See the latter command for more details.

MARK:SID:COUNt[?] command/query

Overlapped: no
Delayed result: no
Pass control required: no
Power-up state: 20

Example Statements:

```
OUTPUT 711;"Mark:B:Sid:Coun 12"
OUTPUT 711;"Marker1:Sideband:Count 154"
OUTPUT 711;"mark:a:sid:coun?"
```

Command Syntax: MARKer[<spec>]:SIDeband:COUNt<sp><value>

<spec>::=~:A~|:B|1|2

<value>::=any integer x, where 0 ≤ x ≤ 200 (NRf format)

Query Syntax: MARKer[<spec>]:SIDeband:COUNt?

Returned Format: <value><LF><^END>

<value>::=an integer (NR1 format)

Description:

Use this command to specify the number of sideband markers you want displayed.

You can either use numbers or one of two nonnumeric parameters to set the value of MARK:SID:COUN. The nonnumeric parameters are:

- UP – increases the value of MARK:SID:COUN by one
- DOWN – decreases the value of MARK:SID:COUN by one

The query returns the number of sideband markers currently specified for the display. The value is returned even if sideband markers are not enabled.

MARK:SID:DELTa[?] command/query

Overlapped: no
Delayed result: no
Pass control required: no
Power-up state: 2048

Example Statements:

```
OUTPUT 711;"MARK:B:SID:DELT 100"
OUTPUT 711;"Marker1:Sideband:Delta 1"
OUTPUT 711;"Mark:A:Sid:Delt?"
```

Command Syntax: MARKer[<spec>]:SIDeband:DELTa<sp><value>[<unit>]

<spec>::=~:A~|:B|1|2

<value>::=any x, where 0 ≤ x ≤ 115 kHz for a one-channel measurement

any x, where 0 ≤ x ≤ 57.5 kHz for a two-channel measurement

Values must be decimal numbers in NRf format.

<unit>::=~HZ~|KHZ

Query Syntax: MARKer[<spec>]:SIDeband:DELTa?

Returned Format: <value><LF><^END>

Description:

Use this command to specify the frequency increment (or delta) between sideband markers.

You can either use numbers or one of three nonnumeric parameters to set the value of MARK:SID:DELT. The nonnumeric parameters are:

- UP – steps MARK:SID:DELT to the next largest acceptable value
- DOWN – steps MARK:SID:DELT to the next smallest acceptable value
- (MARK[:A|:B]:VAL) – sets MARK:SID:DELT to one of two values depending on the marker mode selected. When MARK:X:MODE is NORM, MARK:SID:DELT is set to the value of the main marker. When MARK:X:MODE is DELT, MARK:SID:DELT is set to the difference between the marker reference value and the main marker value.

The query returns the frequency increment currently specified. The value is returned in Hz. It is returned even if sideband markers are not on.

MARK:SID[:FREQuency][?] command/query

Overlapped: no
Delayed result: no
Pass control required: no
Power-up state: 51200

Example Statements:

```
OUTPUT 711;"mark:sid 10khz"
OUTPUT 711;"MARKER:SIDEBAND:FREQUENCY .053KHZ"
OUTPUT 711;"MARK:A:SID?"
```

Command Syntax: MARKer[<spec>]:SIDeband[:FREQuency]<sp><value>[<unit>]

<spec>::=~:A~|:B|1|2

<value>::=any x, where 0 ≤ x ≤ 115 kHz for a one-channel measurement.

any x, where 0 ≤ x ≤ 57.5 kHz for a two-channel measurement.

Values

<unit>::=~HZ~|KHZ

Query Syntax: MARKer[<spec>]:SIDeband[:FREQuency]?

Returned Format: <value><LF><^END>

Description:

This command allows you to specify the carrier frequency for sideband markers and calculations.

You can either use numbers or one of three nonnumeric parameters to set the value of MARK:SID:FREQ. The nonnumeric parameters are:

- UP – steps MARK:SID:FREQ to the value of the next largest point on the x-axis
- DOWN – steps MARK:SID:FREQ to the value of the next smallest point on the x-axis
- (MARK[:A|:B]:VAL) – sets MARK:SID:FREQ to the frequency of the main marker, even when the marker reference is enabled

---

NOTE When you shift the carrier frequency up or down, the sideband markers are all shifted up or down by the same amount.

The query returns the carrier frequency currently being used for sideband markers and calculations. The value is returned in Hz. It is returned even if sideband markers are not enabled.

MARK:SID:POWer? query

Overlapped: no
Delayed result: no
Pass control required: no
Power-up state: 0

Example Statement:

```
OUTPUT 711;"MARK2:SID:POW?"
```

Query Syntax: MARKer[<spec>]:SIDeband:POWer?

<spec>::=~:A~|:B|1|2

Returned Format: <value><LF><^END>

<value>::=a decimal number (NRf format)

Description:

Sideband power can be calculated for the current carrier frequency. This query returns the results of the power calculation.

Sideband power is calculated as an rms summation of the power at all marked sidebands. The result is converted to the current vertical unit (DISP:Y:SCAL:UNIT) after power is calculated. In order for this query to return the results of a power calculation, MARK:SID:STAT and MARK:SID:POW:STAT must be ON.

The query only returns a value. Units are returned with the DISP:Y:SCAL:UNIT query.

MARK:SID:POW:STATe[?] command/query

Overlapped: no
Delayed result: no
Pass control required: no
Power-up state: 1

Example Statements:

```
OUTPUT 711;"Mark:B:Sid:Pow:Stat Off"
OUTPUT 711;"marker1:sideband:power:state 1"
OUTPUT 711;"MARK2:SID:POW:STAT?"
```

Command Syntax: MARKer[<spec>]:SIDeband:POWer:STATe<sp>{OFF|ON|0|1}

<spec>::=~:A~|:B|1|2

Query Syntax: MARKer[<spec>]:SIDeband:POWer:STATe?

Returned Format: {0|1}<LF><^END>

Description:

Use this command to enable and disable the sideband power calculation. When enabled, power is calculated for the marked sidebands and the results of the calculation are displayed on the analyzer's screen. See MARK:SID:POW for more information.

The query returns 0 if sideband power calculation is disabled for the specified display, 1 if it is enabled.

MARK:SID:STATe[?] command/query

Overlapped: no
Delayed result: no
Pass control required: no
Power-up state: 0

Example Statements:
```
OUTPUT 711;"Mark2:Sid:Stat On"
OUTPUT 711;"Marker:A:Sideband:State 0"
OUTPUT 711;"mark:b:sid:stat?"
```

Command Syntax: MARKer[<spec>]:SIDeband:STATe<sp>{OFF|ON|0|1}

<spec>::=~:A~|:B|1|2

Query Syntax: MARKer[<spec>]:SIDeband:STATe?

Returned Format: {0|1}<LF><^END>

Description:

This command enables and disables the sideband markers. Sideband markers must be enabled (MARK:SID:STAT ON) before the results of the sideband power calculation can be returned. See MARK:SID:POW for more information.

At any given time, only one of the following markers can be active in the specified display:

- Band (MARK:BAND:STATE)
- Harmonic (MARK:HARM:STATE)
- Sideband (MARK:SID:STATE)

If you enable the sideband markers when one of the other two is already enabled, that other marker is automatically disabled.

The query returns 0 if the specified display's sideband markers are disabled, 1 if they are enabled.

MARK[:X][?] command/query

Overlapped: no
Delayed result: no
Pass control required: no
Power-up state: 51200 (display A)
0.00390 (display B)

Example Statements:

```
OUTPUT 711;"MARK 100"
OUTPUT 711;"Marker2:X 17KHZ"
OUTPUT 711;"Mark:A:X?"
```

Command Syntax: MARKer[<spec>][:X]<sp><value>[<unit>]

<spec>::=~:A~|:B|1|2

<value>::=a decimal number (NRf format)

<unit>::=~HZ~|KHZ (for frequency domain displays)

~S~|MS|US (for time domain displays)

Query Syntax: MARKer[<spec>][:X]?

Returned Format: <value><LF><^END>

Description:

This command allows you to specify the position of the main marker along the x-axis.

The position is always specified relative to 0 Hz for frequency-domain displays. It is always specified relative to 0 seconds for time-domain displays. You can not specify the position relative to the marker reference, even when MARK:X:MODE is DELT.

You can either use numbers or one of two nonnumeric parameters to set the value of MARK:X. If you use a number, it is rounded to the value of the nearest point on the x-axis. The nonnumeric parameters are:

- UP – steps MARK:X to the value of the next largest point on the x-axis
- DOWN – steps MARK:X to the value of the next smallest point on the x-axis

The query returns the current x-axis value of the main marker if MARK:STAT is ON. The value is returned in Hz for a frequency-domain display and seconds for a time-domain display.

The returned value is never relative to the position of the marker reference, even if MARK:X:MODE is DELT.

MARK[:X]:AMAXimum command

Description:

MARK:X:AMAX is functionally equivalent to MARK:X:AMAX:GLOB. See the latter command for more details.

MARK[:X]:AMAX:AUTO[?] command/query

Overlapped: no
Delayed result: no
Pass control required: no
Power-up state: 0

Example Statements:

```
OUTPUT 711;"mark2:amax:auto off"
OUTPUT 711;"MARKER:A:X:AMAXIMUM:AUTO 1"
OUTPUT 711;"MARK:B:AMAX:AUTO?"
```

Command Syntax: MARKer[<spec>][:X]:AMAXimum:AUTO<sp>{OFF|ON|0|1}

<spec>::=~:A~|:B|1|2

Query Syntax: MARKer[<spec>][:X]:AMAXimum:AUTO?

Returned Format: {0|1}<LF><^END>

Description:

This command enables and disables peak tracking.

When peak tracking is on, the main marker continuously moves to the highest peak on the specified trace. Any peak at 0 Hz is ignored. Peak tracking is automatically turned off when MARK[:X]:AMAX:LEFT or MARK[:X]:AMAX:RIGH is sent to the instrument.

The query returns 0 if peak tracking is disabled for the specified trace, 1 if it is enabled.

MARK[:X]:AMAX[:GLOBal] command

Overlapped: no
Delayed result: no
Pass control required: no
Power-up state: not applicable

Example Statements:
```
OUTPUT 711;"MARK:X:AMAX"
OUTPUT 711;"MARKER:A:X:AMAXIMUM:GLOBAL"
```

Command Syntax: MARKer[<spec>][:X]:AMAXimum[:GLOBal]

<spec>::=~:A~|:B|1|2

Description:

This command moves the main marker to the highest peak on the specified trace.

This command does not allow the main marker to track the highest peak when the trace is updated. Once the peak is found, the marker remains at the point along the x-axis where the peak occurred. If you want the main marker to track the highest peak, use the MARK:X:AMAX:AUTO command.

MARK[:X]:AMAX:LEFT command

Overlapped: no
Delayed result: no
Pass control required: no
Power-up state: not applicable

Example Statements:
```
OUTPUT 711;"Mark1:Amax:Left"
OUTPUT 711;"marker:b:x:amaximum:left"
```

Command Syntax: MARKer[<spec>][:X]:AMAXimum:LEFT

<spec>::=~:A~|:B|1|2

Description:

This command moves the main marker to the next peak to the left of the current marker position.

MARK[:X]:AMAX:RIGHt command

Overlapped: no
Delayed result: no
Pass control required: no
Power-up state: not applicable

Example Statements:
```
OUTPUT 711;"Mark:X:Amax:Righ"
OUTPUT 711;"Marker:A:X:Amaximum:Right"
```

Command Syntax: MARKer[<spec>][:X]:AMAXimum:RIGHt

<spec>::=~:A~|:B|1|2

Description:

This command moves the main marker to the next peak to the right of the current marker position.

MARK[:X]:AMINimum command

Description:

MARK[:X]:AMIN is functionally equivalent to MARK:X:AMIN:GLOB. See the latter command for more details.

MARK[:X]:AMIN[:GLOBal] command

Overlapped: no
Delayed result: no
Pass control required: no
Power-up state: not applicable

Example Statements:
```
OUTPUT 711;"MARK1:X:AMIN:GLOB"
OUTPUT 711;"Marker:B:X:Aminimum:Global"
```

Command Syntax: MARKer[<spec>][:X]:AMINimum[:GLOBal]

<spec>::=~:A~|:B|1|2

Description:

This command moves the main marker to the lowest point on the specified trace.

MARK[:X]:AMPLitude? query

Overlapped: no
Delayed result: no
Pass control required: no
Power-up state: variable

Example Statement: `OUTPUT 711;"mark2:ampl?"`

Query Syntax: MARKer[<spec>][:X]:AMPLitude?

<spec>::=~:A~|:B|1|2

Returned Format: <value><LF><^END>

<value>::=a decimal number (NRf format)

Description:

This query returns one of two values, depending on the setting of MARK:X:MODE.

When MARK:X:MODE is NORM and MARK:X:STAT in ON, the query returns the main marker's is current amplitude. When MARK:X:MODE is DELT and MARK:X:STAT is ON, the query returns the difference between the amplitude of the marker reference and the amplitude of the main marker. Values are returned in the current display unit.

MARK[:X]:AUTO[?] command/query

Overlapped: no
Delayed result: no
Pass control required: no
Power-up state: 0

Example Statements:

```
OUTPUT 711;"MARK:B:X:AUTO 0"
OUTPUT 711;"MARKER1:X:AUTO ON"
OUTPUT 711;"Mark:A:X:Auto?"
```

Command Syntax: MARKer[<spec>][:X]:AUTO<sp>{OFF|ON|0|1}

<spec>::=~:A~|:B|1|2

Query Syntax: MARKer[<spec>][:X]:AUTO?

Returned Format: {0|1}<LF><^END>

Description:

Use this command to enable and disable marker coupling.

Marker coupling links the main markers of display A and display B. When you send a command that moves the main marker of one display, the main marker of the other also moves.

Marker coupling is especially useful when you are displaying the same measurement data using two different trace types. For example, if you were displaying frequency response data using the magnitude and phase trace types, you could track the magnitude and phase of the data at each frequency.

Marker movements are linked via x-axis point number, not by the relative position from the left side of the displays. As a result, linked markers may not always line up on the analyzer's screen. For example, you could display the same measurement data using the same trace type on both displays. But if x-axis points are spaced linearly on display A and logarithmically on display B, the markers will only be aligned on the screen when they are at the first and last x-axis points.

When marker coupling is enabled, the position, of the inactive trace's main marker always corresponds to the position on the active traces main marker. This is true even when peak tracking is enabled (MARK:X:AMAX:AUTO ON) for the inactive trace.

The query returns 0 if marker coupling is disabled, 1 if it is enabled.

MARK[:X]:DELT[?] command/query

Overlapped: no
Delayed result: no
Pass control required: no
Power-up state: 52100 (display A)
0 (display B)

Example Statements:

```
OUTPUT 711;"Mark:A:Delt 100"
OUTPUT 711;"marker:b:x:delta 87khz"
OUTPUT 711;"MARK1:X:DELT?"
```

Command Syntax: MARKer[<spec>][:X]:DELTa<sp><value>[<unit>]

<spec>::=~:A~|:B|1|2

<value>::=a decimal number (NRF format)

<unit>::=~HZ~|KHZ (for frequency domain displays)

~S~|MS|US (for time domain displays)

Query Syntax: MARKer[<spec>][:X]:DELTa?

Returned Format: <value><LF><^END>

<value>::=a decimal number (NRf format)

Description:

A marker reference can be defined by specifying an x-axis and y-axis value. This command allows you to specify the x-axis value of the marker reference. When MARK:X:MODE is DELT, the value returned by MARK:AMPL? is difference between the amplitude of the marker reference and the amplitude of the main marker.

You can either use numbers or one of three nonnumeric parameters to set the value of MARK:X:DELT. The nonnumeric parameters are:

- UP – steps MARK:X:DELT to the value of the next largest point on the x-axis
- DOWN – steps MARK:X:DELT to the value of the next smallest point on the x-axis
- (MARK[:A|:B]:VAL) – sets MARK:X:DELT to the x-axis value of the main marker, even when the marker reference is enabled

To enable the marker reference, MARK:X:STAT must be ON and MARK:X:MODE must be DELT.

The MARK:X:DELT this query returns the current x-axis value of the marker reference. The value is returned in Hz for a frequency-domain display and in seconds for a time-domain display.

MARK[:X]:DELT:AMPLitude[?] command/query

Overlapped: no
Delayed result: no
Pass control required: no
Power-up state: -40 (display A)
0 (display B)

Example Statements:

```
OUTPUT 711;"Mark1:Delt:Ampl 20"
OUTPUT 711;"Marker:X:Delta:Amplitude .5"
OUTPUT 711;"mark2:x:delt:ampl?"
```

Command Syntax: MARKer[<spec>][:X]:DELTa:AMPLitude<sp><value>

<spec>::=~:A~|:B|1|2

<value>::=a decimal number (NRf format)

Query Syntax: MARKer[<spec>][:X]:DELTa:AMPLitude?

Returned Format: <value><LF><^END>

Description:

A marker reference can be defined by specifying an x-axis and y-axis value. This command allows you to specify the y-axis value of the marker reference. When MARK:X:MODE is DELT, the value returned by MARK:X:AMPL? is the difference between the amplitude of the marker reference and the amplitude of the main marker.

You can not specify units for the y-axis value. They are assumed to be the same as the current display units. These can be determined with the DISP:Y:SCAL:UNIT query.

You can either use numbers or one of three nonnumeric parameters to set the value of MARK:X:DELT:AMPL. The nonnumeric parameters are:

- UP – increases the value of MARK:X:DELT:AMPL by half the increment between vertical grid lines
- DOWN – decreases the value of MARK:X:DELT:AMPL by half the increment between vertical grid lines
- (MARK[:A|:B]:VAL) – sets MARK:X:DELT:AMPL to the y-axis value of the main marker, even when the marker reference is enabled

To enable the marker reference, MARK:X:STAT must be ON and MARK:X:MODE must be DELT.

The MARK:X:DELT query returns the current amplitude of the marker reference. The units for this value are returned with the DISP:Y:SCAL:UNIT query.

MARK[:X]:DELT:POINt[?] command/query

Overlapped: no
Delayed result: no
Pass control required: no
Power-up state: 200 (display A)
0 (display B)

Example Statements:

```
OUTPUT 711;"MARK:A:X:DELT:POIN 10"
OUTPUT 711;"Marker2:X:Delta:Point 256"
OUTPUT 711;"Mark:B:X:Delt:Poin?"
```

Command Syntax: MARKer[<spec>][:X]:DELTa:POINt<sp><value>

<spec>::=~:A~|:B|1|2

<value>::=any integer x, where 0 ≤ x ≤ 400 (for a frequency-domain display)

any integer x, where 0 ≤ x ≤ 1023 (for a time-domain display)

Send all values in the NRf format.

Query Syntax: MARKer[<spec>][:X]:DELTa:POINt?

Returned Format: <value><LF><^END>

<value>::=an integer (NR1 format)

Description:

The x-axis is divided into discrete points: 401 points for frequency-domain displays, and 512 or 1024 points for time-domain displays. This command lets you define the x-axis position of the marker reference as a point number rather than a frequency or time.

You can either use numbers or one of two nonnumeric parameters to set the value of MARK:X:DELT:POIN. The nonnumeric parameters are:

- UP – increases the value of MARK:X:DELT:POIN by one
- DOWN – decreases the value of MARK:X:DELT:POIN by one

The query returns the marker reference's x-axis position as a point number.

MARK[:X]:DELT:ZERO — command

Overlapped: no
Delayed result: no
Pass control required: no
Power-up state: not applicable

Example Statements:
```
OUTPUT 711;"MARK:DELT:ZERO"
OUTPUT 711;"Marker1:X:Delta:Zero"
```

Command Syntax: MARKer[<spec>][:X]:DELTa:ZERO

<spec>::=~:A~|:B|1|2

Description:

This command sets the marker reference to the current position of the main marker.

MARK[:X]:MODE[?] — command/query

Overlapped: no
Delayed result: no
Pass control required: no
Power-up state: NORM

Example Statements:
```
OUTPUT 711;"mark:b:mode norm"
OUTPUT 711;"MARKER1:X:MODE DELTA"
OUTPUT 711;"MARK:B:X:MODE?"
```

Command Syntax: MARKer[<spec>][:X]:MODE<sp>{DELTa|NORMal}

<spec>::=~:A~|:B|1|2

Query Syntax: MARKer[<spec>][:X]:MODE?

Returned Format: {DELT|NORM}<LF><^END>

Description:

This command enables and disables the marker reference for the specified display.

When the marker reference is enabled, the main marker's amplitude (returned by MARK:X:AMPL?) is expressed as an amount of offset from the marker reference position. However, the main marker's x-axis position (returned by MARK:X?) is never expressed as an offset from the marker reference, regardless of the setting of MARK:X:MODE.

Use one of the following to define the marker reference position.

- MARK:X:DELT:ZERO
- MARK:X:DELT:AMPL in combination with either MARK:X:DELT or MARK:X:DELT:POIN

The query returns DELT when the marker reference is enabled, NORM when it is disabled.

MARK[:X]:POINt[?] command/query

Overlapped: no
Delayed result: no
Pass control required: no
Power-up state: 200 (display A)
512 (display B)

Example Statements:
```
OUTPUT 711;"MARK:X:POIN 128"
OUTPUT 711;"MARKER:A:X:POINT 512"
OUTPUT 711;"Mark2:X:Poin?"
```

Command Syntax: MARKer[<spec>][:X]:POINt<sp><value>

<spec>::=~:A~|:B|1|2

<value>::=any integer x, where 0 ≤ x ≤ 400 (for a frequency-domain display)

any integer x, where 0 ≤ x ≤ 1023 (for a time-domain display)

Send all values in the NRf format.

Query Syntax: MARKer[<spec>][:X]:POINt?

Returned Format: <value><LF><^END>

<value>::=an integer (NR1 format)

Description:

The x-axis is divided into discrete points: 401 points for frequency-domain displays, and 512 or 1024 points for time-domain displays. This command lets you define the x-axis position of the main marker as a point number rather than a frequency or time.

You can either use numbers or one of two nonnumeric parameters to set the value of MARK:X:POIN. The nonnumeric parameters are:

- UP – increases the value of MARK:X:POIN by one
- DOWN – decreases the value of MARK:X:POIN by one

If MARK:STAT is ON, this query returns the main marker's current x-axis position as a point number.

MARK[:X]:SEARch selector

Description:

This command only selects the MARK:X:SEAR subsystem. Sending MARK:X:SEAR alone does nothing.

MARK[:X]:SEAR:AMPLitude[?] command/query

Overlapped: no
Delayed result: no
Pass control required: no
Power-up state: -3.01 (display A)
1 (display B)

Example Statements:

```
OUTPUT 711;"Mark:B:X:Sear:Ampl 1V"
OUTPUT 711;"marker1:x:search:amplitude 7.2deg"
OUTPUT 711;"MARK:A:X:SEAR:AMPL?"
```

Command Syntax: MARKer[<spec>][:X]:SEARch:AMPLitude<sp><value>[<unit>]

<spec>::=~:A~|:B|1|2

<value>::=a decimal number (NRf format)

<unit>::=any vertical units that are valid for the current display

Query Syntax: MARKer[<spec>][:X]:SEARch:AMPLitude?

Returned Format: <value><LF><^END>

Description:

The analyzer can search for points along the specified trace that intersect a particular y-axis value. Use this command to specify the target value for such a search.

When MARK:X:MODE is NORM, the value you send with this command specifies the target value for the search directly. The search is conducted for points on the trace that intersect the specified y-axis value.

When MARK:X:MODE is DELT, the value you send with this command specifies the target value for the search only indirectly. The value you send with this command is added to the y-axis value of the marker reference to calculate the actual target value. The search is conducted for points on the trace that intersect the calculated y-axis value. This allows you to search for y-axis values that are relative to the marker reference.

You can either use numbers or one of three nonnumeric parameters to set the value of MARK:X:SEAR:AMPL. The nonnumeric parameters are:

- UP – increases the value of MARK:X:SEAR:AMPL by half the increment between y-axis grid lines
- DOWN – decreases the value of MARK:X:SEAR:AMPL by half the increment between y-axis grid lines
- (MARK[:A|:B]:VAL) – sets MARK:X:SEAR:AMPL to one of two values depending on the marker mode selected. When MARK:X:MODE is NORM, MARK:X:SEAR:AMPL is set to the y-axis value of the main marker. When MARK:X:MODE is DELT, MARK:X:SEAR:AMPL is set to the difference between the marker reference value and the main marker value.

Once a target value has been specified, use MARK:X:SEAR:RIGH and MARK:X:SEAR:LEFT to conduct the search.

The query returns the current value of MARK:X:SEAR:AMPL. Units for the value can be returned with the DISP:SCAL:UNIT query.

MARK[:X]:SEAR:LEFT command

Overlapped: no
Delayed result: no
Pass control required: no
Power-up state: not applicable

Example Statements:

```
OUTPUT 711;"Mark1:X:Sear:Left"
OUTPUT 711;"Marker:B:X:Search:Left"
```

Command Syntax: MARKer[<spec>][:X]:SEARch:LEFT

<spec>::=~:A~|:B|1|2

Description:

This command moves the main marker left from its present position to the first occurrence of the y-axis target value. If the target value is not found, the marker is not moved. The target value is specified with the MARK:X:SEAR:AMPL command.

MARK[:X]:SEAR:RIGHt — command

Overlapped: no
Delayed result: no
Pass control required: no
Power-up state: not applicable

Example Statements:
```
OUTPUT 711;"MARK1:X:SEAR:RIGH"
OUTPUT 711;"Marker:B:X:Search:Right"
```

Command Syntax: MARKer[<spec>][:X]:SEARch:RIGHt

<spec>::=~:A~|:B|1|2

Description:

This command moves the marker right from its present position to the first occurrence of the y-axis target value. If the target value is not found, the marker is not moved. The target value is specified with the MARK:X:SEAR:AMPL command.

MARK[:X]:STATe[?] — command/query

Overlapped: no
Delayed result: no
Pass control required: no
Power-up state: 1

Example Statements:
```
OUTPUT 711;"mark2:stat off"
OUTPUT 711;"MARKER:A:X:STATE 1"
OUTPUT 711;"MARK:B:X:STAT?"
```

Command Syntax: MARKer[<spec>][:X]:STATe<sp>{OFF|ON|0|1}

<spec>::=~:A~|:B|1|2

Query Syntax: MARKer[<spec>][:X]:STATe?

Returned Format: {0|1}<LF><^END>

Description:

This command enables and disables the main marker for the specified display.

The query returns 0 if the specified display's main marker is off, 1 if it is on.

MMEMory subsystem

Description:

Commands in this subsystem are used to access the analyzer's mass storage functions (including such things as saving, recalling, and copying files). Many of the commands require a mass storage specifier. The options are:

- RAM: — This specifies a RAM disc that uses some of the analyzer's memory.
- INT: — This specifies the analyzer's internal disc drive.
- EXT: — This specifies an external disc drive connected to the analyzer via the HP-IB. The drive must use the SS/80/HP-IB protocol.

In most cases, if you do not send a mass storage specifier with a command that requires one, a default specifier is assumed. The default specifier is selected with the MMEM:MSI command.

MMEM:COPY command

Overlapped: yes
Delayed result: no
Pass control required: yes only for EXT:
Power-up state: not applicable

NOTE Do not use this command to copy all files at once if the destination device contains files you want to keep. All files on the destination device are overwritten when you specify an all-file copy.

Example Statements:

```
OUTPUT 711;"MMEM:COPY 'RAM:','INT:'"
OUTPUT 711;"MMEMORY:COPY ""RAM:MyFile"","EXT:MyExtFile"""
```

Command Syntax: MMEMory:COPY<sp>{'|"}<source>{'|"},{'|"}<destination>{'|"}

<source>::=EXT:|INT:|RAM: (when copying the entire contents of one mass storage device to another)

[EXT:|INT:|RAM:]<filename> (when copying a single file)

<source> designates the mass storage device or file that will be copied.

<destination>::=EXT:|INT:|RAM: (when copying the entire contents of one mass storage device to another)

[EXT:|INT:|RAM:]<filename> (when copying a single file)

<destination> designates the mass storage device or file that will receive the new copy

<filename>::=1 to 10 printable ASCII characters

Description:

Use this command to copy files. You can copy files one at a time or you can copy all of the files on one mass storage device to another device.

To copy one file, the name of the file you want to copy should be entered as the <source>. The name for the new file should be entered as the <destination>. You must precede the source filename with a mass storage specifier unless the file resides on the default mass storage device. You must precede the destination filename with a mass storage specifier unless you want the new copy to be placed on the default mass storage device. (Use the MMEM:MSI query to determine the default device.)

To copy all files on one mass storage device to another device, the device containing the files should be entered as the <source>. The device that will receive the new copies should be entered as the <destination>.

When execution of this command requires access to the external mass storage device (EXT:), the active controller on the HP-IB must temporarily pass control to the analyzer. When execution of the command has been completed, the analyzer must pass control back. For more information on passing control, see Chapter 2, "Behavior in an HP-IB System."

MMEM:DELete command

Overlapped: yes only for EXT:
Delayed result: no
Pass control required: yes only for EXT:
Power-up state: not applicable

Example Statements:

```
OUTPUT 711;"Mmem:Del 'RAM:'"
OUTPUT 711;"mmemory:delete 'INT:myspec'"
```

Command Syntax: MMEMory:DELete<sp>{'|"}<target>{'|"}

<target>::=EXT:|INT:|RAM: (when deleting everything from a mass storage device)

[EXT:|INT:|RAM:]<filename> (when deleting one file from a mass storage device)

<filename>::=1 to 10 printable ASCII characters

Description:

Use this command to delete either one file or all files from a mass storage device.

To delete just one file, enter the filename as the <target>. You must precede the filename with a mass storage specifier unless the file resides on the default mass storage device. (Use the MMEM:MSI query to determine the default device.)

To delete all files from a mass storage device, enter the device specifier as the <target>.

When execution of this command requires access to the external mass storage device (EXT:), the active controller on the HP-IB must temporarily pass control to the analyzer. When execution of the command has been completed, the analyzer must pass control back. For more information on passing control, see Chapter 2, "Behavior in an HP-IB System."

MMEM:FORM[?] command/query

Overlapped: no
Delayed result: no
Pass control required: no
Power-up state: BIN

Example Statements:

```
OUTPUT 711;"Mmem:Form Bin"
OUTPUT 711;"Mmemory:Form Ascii"
OUTPUT 711;"mmem:form?"
```

Command Syntax: MMEMory:FORM<sp>{ASCii|BINary}

Query Syntax: MMEMory:FORM?

Returned Format: {ASC|BIN}<LF><^END>

Description:

This command selects the type of data encoding that will be used when files are saved. The two types of encoding are ASCII and binary.

The option you select here is only needed when a file is saved using the analyzer's front-panel keys. This is because a data encoding type is already implied in the HP-IB commands used to save files. For example, to save a trace into a binary-encoded file, use the HP-IB command:

MMEM:STOR:TRAC 'trace_1'

To save the same trace into an ASCII-encoded file, use the HP-IB command:

MMEM:SAVE:TRAC 'trace_1'

The query returns ASC or BIN depending on the option last selected.

MMEM:GET selector

Description:

This command only selects the MMEM:GET subsystem. Sending MMEM:GET alone does nothing.

Commands in the MMEM:GET subsystem are used to load either trace or setup information into the analyzer from files on a mass storage device. All of the commands allow you to specify the device on which files reside. If you do not specify a device, however, the default device is assumed. (Use the MMEM:MSI query to determine the default device.)

When execution of a command requires access to the external mass storage device (EXT:), the active controller on the HP-IB must temporarily pass control to the analyzer. When execution of the command has been completed, the analyzer must pass control back. For more information on passing control, see Chapter 2, "Behavior in an HP-IB System."

There is no difference between comparable commands in the two subsystems MMEM:GET and MMEM:LOAD subsystems. Whether the file is stored as ASCII or binary, a GET or LOAD will read it in.

MMEM:GET:DTABle command

Overlapped: yes only for EXT:
Delayed result: no
Pass control required: yes only for EXT:
Power-up state: not applicable

Example Statements:

```
OUTPUT 711;"MMEM:GET:DTAB1 'INT:MYTABLE'"
OUTPUT 711;"Mmemory:Get:Dtable:B ""EXT:myTable"""
```

Command Syntax: MMEMory:GET:DTABle[<spec>]<sp>{'|"}[<msi>]<filename>{'|"}

<spec>::=~:A~|:B|1|2

<msi>::=EXT:|INT:|RAM:

<filename>::=name of the file you want to load (the file must contain a data table)

Description:

Use this command to recall a data table from a file. The data table will be coupled to the display specified in <spec>.

See MMEM:GET for more information.

MMEM:GET:LIMit command

Overlapped: yes only for EXT:
Delayed result: no
Pass control required: yes only for EXT:
Power-up state: not applicable

Example Statements:

```
OUTPUT 711;"mmem:get:lim3 ""RAM:mylimit"""
OUTPUT 711;"MMEMORY:GET:LIMIT5 'INT:YOURLIMIT'"
```

Command Syntax: MMEMory:GET:LIMit<spec><sp>{'|"}[<msi>]<filename>{'|"}

<spec>::=a single integer from 1 to 8

<msi>::=EXT:|INT:|RAM:

<filename>::=name of the file you want to load (the file must contain a limit table)

Description:

Use this command to recall a limit table from a file. Because the analyzer has places for eight limit tables, you must use <spec> to indicate which table should receive the file.

See MMEM:GET for more information.

MMEM:GET:MATH command

Overlapped: yes only for EXT:
Delayed result: no
Pass control required: yes only for EXT:
Power-up state: not applicable

Example Statements:

```
OUTPUT 711;"MMEM:GET:MATH 'EXT:MYMATH'"
OUTPUT 711;"MMEMORY:GET:MATH 'INT:Definition'"
```

Command Syntax: MMEMory:GET:MATH<sp>{'|"}[<msi>]<filename>{'|"}

<msi>::=EXT:|INT:|RAM:

<filename>::=name of the file you want to load (the file must contain math definitions)

Description:

Use this command to recall math definitions from a file.

The analyzer allows you to define five math functions and five constants. When you save or recall a math file, you are saving or recalling all five function and constant definitions at once. You can not save or recall individual functions or constants.

See MMEM:GET for more information.

MMEM:GET:STATe command

Overlapped: yes only for EXT:
Delayed result: no
Pass control required: yes only for EXT:
Power-up state: not applicable

Example Statements:

```
OUTPUT 711;"Mmem:Get:Stat ""RAM:MYSTATE"""
OUTPUT 711;"mmemory:get:state ""EXT:MyFile"""
```

Command Syntax: MMEMory:GET:STATe<sp>{'|"}[<msi>]<filename>{'|"}

<msi>::=EXT:|INT:|RAM:

<filename>::=name of the file you want to load (the file must contain an instrument state)

Description:

Use this command to recall an instrument state (setup) from a file.

NOTE In addition to setup information, instrument-state files include definitions for the following items: all eight limit tables, both data tables, and all five math functions and constants. As a result, the current definitions of these items are all overwritten when you recall an instrument state.

Some of your measurements may require analyzer setups that are significantly different from the preset state. If you make these measurements often, you may want to save the special setups in instrument-state files. When you change from one setup to another, you can save time by recalling an instrument state rather than sending many individual commands.

See MMEM:GET for more information.

MMEM:GET:TRACe command

Overlapped: yes only for EXT:
Delayed result: no
Pass control required: yes only for EXT:
Power-up state: not applicable

Example Statements:
```
OUTPUT 711;"Mmem:Get:Trac 'INT:tracetype'"
OUTPUT 711;"Mmemory:Get:Trace1 ""EXT:MYTRACE"""
```

Command Syntax: MMEMory:GET:TRACe[<spec>]<sp>{'|"}[<msi>]<filename>{'|"}

<spec>::=~:A~|:B|1|2

<msi>::=EXT:|INT:|RAM:

<filename>::=name of the file you want to load (the file must contain a trace)

Description:

Use this command to recall a trace from a file. The trace will be placed into the display specified in <spec>.

See MMEM:GET for more information.

MMEM:INITialize command

Overlapped: yes
Delayed result: no
Pass control required: yes only for EXT:
Power-up state: not applicable

Example Statements:
```
OUTPUT 711;"MMEM:INIT 'INT:MY_DIR'"
OUTPUT 711;"Mmemory:Initialize 'RAM:'"
```

Command Syntax: MMEMory:INITialize<sp>{'|"}<msi>[<volume_name>]{'|"}

<msi>::=EXT:|INT:|RAM:

<volume_name>::=1 to 10 printable ASCII characters

Description:

Use this command to format the specified mass storage device. Formatting proceeds according to selections last made with the MMEM:INIT:INT and MMEM:INIT:OPT commands.

When you format a device, you can give it a volume name. This is especially useful for identifying the floppy discs you use in the analyzer's internal disc drive. The name is displayed on the analyzer's screen when the catalog is turned on (SCR:CONT DCAT).

When execution of this command requires access to the external mass storage device (EXT:), the active controller on the HP-IB must temporarily pass control to the analyzer. When execution of the command has been completed, the analyzer must pass control back. For more information on passing control, see Chapter 2, "Behavior in an HP-IB System."

MMEM:INIT:INTerleave[?] command/query

Overlapped: no
Delayed result: no
Pass control required: no
Power-up state: 1

Example Statements:

```
OUTPUT 711;"mmem:init:int 1"
OUTPUT 711;"MMEMORY:INITIALIZE:INTERLEAVE 5"
OUTPUT 711;"MMEM:INIT:INT?"
```

Command Syntax: MMEMory:INITialize:INTerleave<sp><factor>

<factor>::=an integer from 1 through 255 (NRf format)

Query Syntax: MMEMory:INITialize:INTerleave?

Returned Format: <factor><LF><^END>

<factor>::=an integer (NR1 format)

Description:

This command lets you specify an interleave factor to be used when analyzer formats a mass storage device.

During formatting, each track on a disc is divided into sectors, and the sectors are numbered in a pattern determined by the interleave factor. The numbering pattern has an effect on the efficiency of disc-read and disc-write operations.

For the analyzer's internal disc drive, an interleave factor of 1 results in the most efficient disc operations. If you use an external disc drive, you will need to check the drive's documentation to determine which interleave factor will work best.

You can either use numbers or one of two nonnumeric parameter to set the value of MMEM:INIT:INT. The nonnumeric parameters are:

- UP – increases the current value of MMEM:INIT:INT by one
- DOWN – decreases the current value of MMEM:INIT:INT by one

The query tells you which disc interleave factor is currently selected.

MMEM:INIT:OPTion[?] command/query

Overlapped: no
Delayed result: no
Pass control required: no
Power-up state: 0

Example Statements:
```
OUTPUT 711;"MMEM:INIT:OPT 64"
OUTPUT 711;"MMEMory:INITialize:OPTion 256"
OUTPUT 711;"MMEM:INIT:OPT?"
```

Command Syntax: MMEMory:INITialize:OPTion<sp><value>

<value>::=an integer from 0 through 16777216 (NRf format)

Query Syntax: MMEMory:INITialize:OPTion?

Returned Format: <value><LF><^END>

<value>::=an integer (NR1 format)

Description:

This command lets you specify the format option to be used when the analyzer formats a mass storage device.

The format option is an encoded value whose meaning is dependent on the mass storage device being formatted. If you are formatting an external disc drive, you must refer to its documentation to decode the different format option values.

Format options 0 through 5 are used to allocate memory for the analyzer's RAM disc (RAM:). They are also used to allocate disc space on the analyzer's internal disc drive. The amount of memory or disc space (in bytes) for these options are as follows:

Option	RAM Disc	Internal Disc
0	64k	640k
1	640k	640k
2	710k	710k
3	788k	788k
4	270k	– –
5	640k	64k

When formatting the RAM disc, you can also use the MMEM:INIT:OPT command to directly specify the amount of memory you want allocated for the RAM disc. In this case, <value> is not encoded. It should contain the amount of memory you want allocated (in bytes). The amount you specify is rounded up to the nearest multiple of 256 bytes. You can determine how much memory is available for the RAM disc by sending SCR:CONT MEM and examining the information displayed on the analyzer's screen.

The query response tells you which option is currently selected.

MMEM:LOAD selector

Description:

This command only selects the MMEM:LOAD subsystem. Sending MMEM:LOAD alone does nothing.

Commands in the MMEM:LOAD subsystem are used to load either trace or setup information into the analyzer from files on a mass storage device. All of the commands allow you to specify the device on which files reside. If you do not specify a device, however, the default device is assumed. (Use the MMEM:MSI query to determine the default device.)

When execution of a command requires access to the external mass storage device (EXT:), the active controller on the HP-IB must temporarily pass control to the analyzer. When execution of the command has been completed, the analyzer must pass control back. For more information on passing control, see Chapter 2, "Behavior in an HP-IB System."

There is no difference between comparable commands in the MMEM:LOAD and MMEM:GET subsystems.

MMEM:LOAD:APPLication command

Overlapped: yes only for EXT:
Delayed result: no
Pass control required: yes only for EXT:
Power-up state: not applicable

Example Statements:

```
OUTPUT 711;"MMEM:LOAD:APPL ""INT:myappl1"""
OUTPUT 711;"MMEMORY:LOAD:APPLICATION 'EXT:MyAppl2'"
```

Command Syntax: MMEMory:LOAD:APPLication<sp>{'|"}[<msi>]<filename>{'|"}

<msi>::=EXT:|INT:|RAM:

<filename>::=name of the file you want to load (the file must contain an application program written for the HP 35660A)

Description:

Use this command to install an application from a file.

NOTE Applications that run on the HP 35660A may add some HP-IB commands that are not described in this manual. See the application's documentation for more information.

See MMEM:LOAD for more information.

MMEM:LOAD:APPL:ALL command

Overlapped: yes only for EXT:
Delayed result: no
Pass control required: yes only for EXT:
Power-up state: not applicable

Example Statements:

```
OUTPUT 711;"Mmem:Load:Appl:All ""INT:"""
OUTPUT 711;"MMEMORY:LOAD:APPLICATION:ALL 'EXT:'"
```

Command Syntax: MMEMory:LOAD:APPLication:ALL<sp>{'|"}{EXT:|INT:|RAM:}{'|"}

Description:

Use this command to install all applications that reside on the specified mass storage device.

NOTE Applications that run on the HP 35660A may add some HP-IB commands that are not described in this manual. See the application's documentation for more information.

See MMEM:LOAD for more information.

MMEM:LOAD:APPL:AUTO[?] command/query

Overlapped: no
Delayed result: no
Pass control required: no
Power-up state: saved in nonvolatile memory

Example Statements:

```
OUTPUT 711;"MMEM:LOAD:APPL:AUTO ON"
OUTPUT 711;"Mmemory:Load:Application:Auto 0"
OUTPUT 711;"mmem:load:appl:auto?"
```

Command Syntax: MMEMory:LOAD:APPLication:AUTO<sp>{OFF|ON|0|1}

Query Syntax: MMEMory:LOAD:APPLication:AUTO?

Returned Format: {0|1}<LF><^END>

Description:

This command allows you to specify whether or not applications should be loaded automatically when you turn the analyzer on.

If MMEM:LOAD:APPL:AUTO is ON, any application whose name ends with "_LD" is automatically loaded into the analyzer at power-up. First, the analyzer loads all such applications from the internal disc drive. Then, if the analyzer is the system controller and an external disc drive is connected to the HP-IB, the analyzer loads all such applications from the external drive.

The option last specified with this command is saved in nonvolatile memory when you send the SYST:SAVE command. This means that when you turn the analyzer off and then back on, the state of MMEM:LOAD:APPL:AUTO does not change.

MMEM:LOAD:DTABle command

Overlapped: yes only for EXT:
Delayed result: no
Pass control required: yes only for EXT:
Power-up state: not applicable

Example Statements:
```
OUTPUT 711;"Mmem:Load:Dtab2 ""RAM:MYTABLE4"""
OUTPUT 711;"mmemory:load:dtable:a 'EXT:dataTable'"
```

Command Syntax: MMEMory:LOAD:DTABle[<spec>]<sp>{'|"}[<msi>]<filename>{'|"}

<spec>::=~:A~|:B|1|2

<msi>::=EXT:|INT:|RAM:

<filename>::=name of the file you want to load (the file must contain a data table)

Description:

Use this command to recall a data table from a file. The data table will be coupled to the display specified in <spec>.

See MMEM:LOAD for more information.

MMEM:LOAD:LIMit command

Overlapped: yes only for EXT:
Delayed result: no
Pass control required: yes only for EXT:
Power-up state: not applicable

Example Statements:
```
OUTPUT 711;"Mmem:Load:Liml 'INT:MYLIMIT'"
OUTPUT 711;"Mmemory:Load:Limit7 ""EXT:extlimit"""
```

Command Syntax: MMEMory:LOAD:LIMit<spec><sp>{'|"}[<msi>]<filename>{'|"}

<spec>::=a single integer from 1 to 8

<msi>::=EXT:|INT:|RAM:

<filename>::=name of the file you want to load (the file must contain a limit table)

Description:

Use this command to recall a limit table from a file. Because the analyzer has places for eight limit tables, you must use <spec> to indicate which table should receive the file.

See MMEM:LOAD for more information.

MMEM:LOAD:MATH command

Overlapped: yes only for EXT:
Delayed result: no
Pass control required: yes only for EXT:
Power-up state: not applicable

Example Statements:

```
OUTPUT 711;"MMEM:LOAD:MATH 'RAM:myMath'"
OUTPUT 711;"Mmemory:Load:Math 'INT:mathdef'"
```

Command Syntax: MMEMory:LOAD:MATH<sp>{'|"}[<msi>]<filename>{'|"}

<msi>::=EXT:|INT:|RAM:

<filename>::=name of the file you want to load (the file must contain math definitions)

Description:

Use this command to recall math definitions from a file.

The analyzer allows you to define five math functions and five constants. When you save or recall a math file, you are saving or recalling all five function and constant definitions at once. You can not save or recall individual functions or constants.

See MMEM:LOAD for more information.

MMEM:LOAD:STATe command

Overlapped: yes only for EXT:
Delayed result: no
Pass control required: yes only for EXT:
Power-up state: not applicable

Example Statements:

```
OUTPUT 711;"mmem:load:stat ""RAM:MyState"""
OUTPUT 711;"MMEMORY:LOAD:STATE ""EXT:INST_STATE"""
```

Command Syntax: MMEMory:LOAD:STATe<sp>{'|"}[<msi>]<filename>{'|"}

<msi>::=EXT:|INT:|RAM:

<filename>::=name of the file you want to load (the file must contain an instrument state)

Description:

Use this command to recall an instrument state (setup) from a file.

NOTE In addition to setup information, instrument-state files include definitions for the following items: all eight limit tables, both data tables, and all five math functions and constants. As a result, the current definitions of these items are all overwritten when you recall an instrument state.

Some of your measurements may require analyzer setups that are significantly different from the preset state. If you make these measurements often, you may want to save the special setups in instrument-state files. When you change from one setup to another, you can save time by recalling an instrument state rather than sending many individual commands.

See MMEM:LOAD for more information.

MMEM:LOAD:TRACe command

Overlapped: yes only for EXT:
Delayed result: no
Pass control required: yes only for EXT:
Power-up state: not applicable

Example Statements:

```
OUTPUT 711;"MMEM:LOAD:TRAC ""INT:A_TRACE"""
OUTPUT 711;"MMEMORY:LOAD:TRACE 'RAM:mytrace'"
```

Command Syntax: MMEMory:LOAD:TRACe[<spec>]<sp>{'|"}[<msi>]<filename>{'|"}

<spec>::=~:A~|:B|1|2

<msi>::=EXT:|INT:|RAM:

<filename>::=name of the file you want to load (the file must contain a trace)

Description:

Use this command to recall a trace from a file. The trace will be called into the display specified in <spec>.

See MMEM:LOAD for more information.

MMEM:MSI[?] command/query

Overlapped: no
Delayed result: no
Pass control required: no
Power-up state: saved in nonvolatile memory

Example Statements:

```
OUTPUT 711;"Mmem:Msi 'RAM:'"
OUTPUT 711;"mmemory:msi ""INT:"""
OUTPUT 711;"MMEM:MSI?"
```

Command Syntax: MMEMory:MSI<sp>{'|"}{EXT:|INT:|RAM:}{'|"}

Query Syntax: MMEMory:MSI?

Returned Format: "{EXT:|INT:|RAM:}"<LF><^END>

Description:

This command allows you to indicate which of the three mass storage devices will be the default device.

Many commands in the MMEMory subsystem allow you to either include or omit a mass storage specifier. When you omit the specifier, the default device is used.

The option last specified with this command is saved in nonvolatile memory when you send the SYST:SAVE command. This means that when you turn the analyzer off and then back on, the state of MMEM:MSI does not change.

The query tells you which mass storage device is currently specified as the default.

MMEM:MSI:ADDRess[?] command/query

Overlapped: no
Delayed result: no
Pass control required: no
Power-up state: saved in nonvolatile memory

Example Statements:

```
OUTPUT 711;"Mmem:Msi:Addr 1"
OUTPUT 711;"Mmemory:Msi:Address 6"
OUTPUT 711;"mmem:msi:addr?"
```

Command Syntax: MMEMory:MSI:ADDRess<sp><value>

<value>::=any integer x, where 0 ≤ x ≤ 7 (NRf format)

Query Syntax: MMEMory:MSI:ADDRess?

Returned Format: <value><LF><^END>

<value>::=an integer (NR1 format)

Description:

Use this command to enter the address of an external disc drive (EXT:) connected to the analyzer's HP-IB. What you enter here must match the address setting of the drive's HP-IB address switches. (Refer to the disc drive's documentation for the location of its address switches.)

You can either use numbers or one of two nonnumeric parameters to set the value of MMEM:MSI:ADDR. The nonnumeric parameters are:

- UP – increases the current value of MMEM:MSI:ADDR by one
- DOWN – decreases the current value of MMEM:MSI:ADDR by one

The option last specified with this command is saved in nonvolatile memory when you send the SYST:SAVE command. This means that when you turn the analyzer off and then back on, the state of MMEM:MSI:ADDR does not change.

The query returns the HP-IB address at which the analyzer expects to find the external disc drive.

MMEM:MSI:UNIT[?] command/query

Overlapped: no
Delayed result: no
Pass control required: no
Power-up state: saved in nonvolatile memory

Example Statements:

```
OUTPUT 711;"MMEM:MSI:UNIT 1"
OUTPUT 711;"Mmemory:Msi:Unit 7"
OUTPUT 711;"Mmem:Msi:Unit?"
```

Command Syntax: MMEMory:MSI:UNIT<sp><value>

<value>::=any integer x, where 0 ≤ x ≤ 15 (NRf format)

Query Syntax: MMEMory:MSI:UNIT?

Returned Format: <value><LF><^END>

<value>::=an integer (NR1 format)

Description:

This command allows you to indicate which unit of the external disc drive (EXT:) should be used for mass storage.

If the external drive has one HP-IB port for more than one mass storage unit, each unit is specified by a different number. For disc drives with only one mass storage unit, the unit number is 0 (zero). See the disc drive's documentation for more information on determining the unit number.

You can either use numbers or one of two nonnumeric parameters to set the value of MMEM:MSI:UNIT. The nonnumeric parameters are:

- UP – increases the current value of MMEM:MSI:UNIT by one
- DOWN – decreases the current value of MMEM:MSI:UNIT by one

The unit number last specified with this command is saved in nonvolatile memory when you send the SYST:SAVE command. This means that when you turn the analyzer off and then back on, the number does not change.

The query returns the unit number currently specified for the external disc drive.

MMEM:MSI:VOLume[?] command/query

Overlapped: no
Delayed result: no
Pass control required: no
Power-up state: saved in nonvolatile memory

Example Statements:

```
OUTPUT 711;"mmem:msi:vol 2"
OUTPUT 711;"MMEMORY:MSI:VOLUME 5"
OUTPUT 711;"MMEM:MSI:VOL?"
```

Command Syntax: MMEMory:MSI:VOLume<sp><value>

<value>::=any integer x, where $0 \leq x \leq 7$ (NRf format)

Query Syntax: MMEMory:MSI:VOLume?

Returned Format: <value><LF><^END>

<value>::=integer in NR1 format.

Description:

This command allows you to specify which volume of the external disc drive (EXT:) should be used for mass storage.

If the external drive contains multiple volumes, each is specified by a different number. If the drive contains only one volume, the volume number is 0 (zero). See the disc drive's documentation for more information on determining the volume number.

You can either use numbers or one of two nonnumeric parameters to set the value of MMEM:MSI:VOL. The nonnumeric parameters are:

- UP – increases the current value of MMEM:MSI:VOL by one
- DOWN – decreases the current value of MMEM:MSI:VOL by one

The volume number last specified with this command is saved in nonvolatile memory when you send the SYST:SAVE command. This means that when you turn the analyzer off and then back on, the number does not change.

The query returns the volume number currently specified for the external disc drive.

MMEM:PACK command

Overlapped: yes only for EXT:
Delayed result: no
Pass control required: yes only for EXT:
Power-up state: not applicable

Example Statements:

```
OUTPUT 711;"Mmem:Pack ""RAM:"""
OUTPUT 711;"mmemory:pack 'EXT:'"
```

Command Syntax: MMEMory:PACK<sp>{'|"}{EXT:|INT:|RAM:}{'|"}

Description:

Use this command to recover unused space between files on the specified mass storage device.

Spaces can be left between the files on a mass storage device when other files are deleted from the device. These spaces may be inaccessible when you are storing new files.

When you send this command, all files are shifted toward the beginning of the device's memory or disc space so that they are directly adjacent to one another. This shifts the space between those files to the end of memory or disc space. The recovered space can then be used for storing new files.

When execution of this command requires access to the external mass storage device (EXT:), the active controller on the HP-IB must temporarily pass control to the analyzer. When execution of the command has been completed, the analyzer must pass control back. For more information on passing control, see Chapter 2, "Behavior in an HP-IB System."

MMEM:RENname command

Overlapped: yes
Delayed result: no
Pass control required: yes only for EXT:
Power-up state: not applicable

Example Statements:

```
OUTPUT 711;"MMEM:REN 'INT:MYFILE','INT:MyFile'"
OUTPUT 711;"MMEMORY:RENAME ""EXT:VOLUME2"","EXT:VOLUME4"""
```

Command Syntax: MMEMory:RENName<sp>{'|"}<old_name>{'|"},{'|"}<new_name>{'|"}

<old_name>::=[<msi>]<filename> (when changing the name of a file)

<msi> (when changing the name of a device volume)

<old_name> designates the mass storage device or the file you want to rename

<new_name>::=[<msi>]<filename> (when changing the name of a file)

[<msi>]<volume_name> (when changing the name of a device volume)

<new_name> designates the new name of the file or mass storage device

<msi>::=EXT:|INT:|RAM:

<msi> must be the same for both <old_name> and <new_name>.

<filename>::=1 to 10 printable ASCII characters

<volume_name>::=1 to 10 printable ASCII characters

Description:

This command allows you to change the volume name of a mass storage device or to change the name of a file.

Volume names can be especially useful for identifying the floppy discs you use in the analyzer's internal disc drive. The name is displayed on the analyzer's screen when the catalog is turned on (SCR:CONT DCAT).

When execution of this command requires access to the external mass storage device (EXT:), the active controller on the HP-IB must temporarily pass control to the analyzer. When execution of the command has been completed, the analyzer passes control back. For more information on passing control, see Chapter 2, "Behavior in an HP-IB System."

MMEM:SAVE — selector

Description:

This command only selects the MMEM:SAVE subsystem. Sending MMEM:SAVE alone does nothing.

Commands in the MMEM:SAVE subsystem are used to save the analyzer's trace and setup information to files on a mass storage device. All of the commands allow you to specify the device on which files will be saved. If you do not specify a device, however, the default device is assumed. (Use the MMEM:MSI query to determine the default device.)

When execution of a command requires access to the external mass storage device (EXT:), the active controller on the HP-IB must temporarily pass control to the analyzer. When execution of the command has been completed, the analyzer passes control back. For more information on passing control, see Chapter 2, "Behavior in an HP-IB System."

The MMEM:SAVE and MMEM:STOR subsystems are very similar. Commands in both subsystems let you save files. But there is one major difference: files saved with SAVE commands will be ASCII encoded while files saved with STOR commands will be binary encoded. The difference between these two kinds of encoding is discussed in Chapter 4, "Transferring Data."

MMEM:SAVE:DTABle — command

Overlapped: yes only for EXT:
Delayed result: no
Pass control required: yes only for EXT:
Power-up state: not applicable

Example Statements:

```
OUTPUT 711;"Mmem:Save:Dtab:A ""EXT:mytable"""
OUTPUT 711;"Mmemory:Save:Dtable2 'INT:DATA_FILE2'"
```

Command Syntax:

MMEMory:SAVE:DTABle[<spec>]<sp>{'|"}[<msi>]<filename>{'|"}

<spec>::=~:A~|:B|1|2

<msi>::=EXT:|INT:|RAM:

<filename>::=1 to 10 printable ASCII characters

Description:

Use this command to save a data table into an ASCII-encoded file. The data table is saved from the display specified in <spec>.

See MMEM:SAVE for more information.

MMEM:SAVE:LIMit command

Overlapped: yes only for EXT:
Delayed result: no
Pass control required: yes only for EXT:
Power-up state: not applicable

Example Statements:
```
OUTPUT 711;"MMEM:SAVE:LIM4 'RAM:LIM_TABLE1'"
OUTPUT 711;"Mmemory:Save:Limit8 'INT:MYLIMIT'"
```

Command Syntax: MMEMory:SAVE:LIMit<spec><sp>{'|"}[<msi>]<filename>{'|"}

<spec>::=a single integer from 1 to 8

<msi>::=EXT:|INT:|RAM:

<filename>::=1 to 10 printable ASCII characters

Description:

Use this command to save a limit table into an ASCII-encoded file. The data table saved is the one specified in <spec>.

See MMEM:SAVE for more information.

MMEM:SAVE:MATH command

Overlapped: yes only for EXT:
Delayed result: no
Pass control required: yes only for EXT:
Power-up state: not applicable

Example Statements:
```
OUTPUT 711;"mmem:save:math ""EXT:math_def1"""
OUTPUT 711;"MMEMORY:SAVE:MATH ""INT:MathFile4"""
```

Command Syntax: MMEMory:SAVE:MATH<sp>{'|"}[<msi>]<filename>{'|"}

<msi>::=EXT:|INT:|RAM:

<filename>::=1 to 10 printable ASCII characters

Description:

Use this command to save math definitions into an ASCII-encoded file.

The analyzer allows you to define five math functions and five constants. When you save or recall a math file, you are saving or recalling all five function and constant definitions at once. You can not save or recall individual functions or constants.

See MMEM:SAVE for more information.

MMEM:SAVE:STATe command

Overlapped: yes only for EXT:
Delayed result: no
Pass control required: yes only for EXT:
Power-up state: not applicable

Example Statements:

```
OUTPUT 711;"MMEM:SAVE:STAT 'INT:Instr_4'"
OUTPUT 711;"MMEMORY:SAVE:STATE ""EXT:myState"""
```

Command Syntax: MMEMory:SAVE:STATe<sp>{'|"}[<msi>]<filename>{'|"}

<msi>::=EXT:|INT:|RAM:

<filename>::=1 to 10 printable ASCII characters

Description:

Use this command to save the current instrument state (setup) into an ASCII-encoded file.

NOTE When you save an analyzer setup, you are also saving the following items: all eight limit tables, both data tables, and all five math functions and constants. To decrease the size of an instrument-state file, you can save limit and data tables separately and then clear them before saving the setup.

Some of your measurements may require analyzer setups that are significantly different from the preset state. If you make these measurements often, you may want to save the special setups in instrument-state files. When you change from one setup to another, you can save time by recalling an instrument state rather than sending many individual commands.

See MMEM:SAVE for more information.

MMEM:SAVE:TRACe command

Overlapped: yes only for EXT:
Delayed result: no
Pass control required: yes only for EXT:
Power-up state: not applicable

Example Statements:
```
OUTPUT 711;"Mmem:Save:Tracl ""EXT:trace7"""
OUTPUT 711;"mmemory:save:trace:b 'INT:B_Trace'"
```

Command Syntax: MMEMory:SAVE:TRACe[<spec>]<sp>{'|"}[<msi>]<filename>{'|"}

<spec>::=~:A~|:B|1|2

<msi>::=EXT:|INT:|RAM:

<filename>::=1 to 10 printable ASCII characters

Description:

Use this command to save a trace into an ASCII-encoded file. The trace will be saved from the display specified in <spec>.

See MMEM:SAVE for more information.

MMEM:STORe selector

Description:

This command only selects the MMEM:STOR subsystem. Sending MMEM:STOR alone does nothing.

Commands in the MMEM:STOR subsystem are used to save the analyzer's trace and setup information to files on a mass storage device. All of the commands allow you to specify the device on which files will be saved. If you do not specify a device, however, the default device is assumed. (Use the MMEM:MSI query to determine the default device.)

When execution of a command requires access to the external mass storage device (EXT:), the active controller on the HP-IB must temporarily pass control to the analyzer. When execution of the command has been completed, the analyzer passes control back. For more information on passing control, see Chapter 2, "Behavior in an HP-IB System."

The MMEM:STOR and MMEM:SAVE subsystems are very similar. Commands in both subsystems let you save files. But there is one major difference: files saved with STOR commands will be binary encoded while files saved with SAVE commands will be ASCII encoded. The difference between these two kinds of encoding is discussed in Chapter 4, "Transferring Data."

MMEM:STOR:DTABle command

Overlapped: yes only for EXT:
Delayed result: no
Pass control required: yes only for EXT:
Power-up state: not applicable

Example Statements:

```
OUTPUT 711;"Mmem:Stor:Dtab:B 'INT:mytable'"
OUTPUT 711;"Mmemory:Store:Dtable2 'RAM:TABLE_9'"
```

Command Syntax: MMEMory:STORe:DTABle[<spec>]<sp>{'|"}[<msi>]<filename>{'|"}

<spec>::=~:A~|:B|1|2

<msi>::=EXT:|INT:|RAM:

<filename>::=1 to 10 printable ASCII characters

Description:

Use this command to save a data table into a binary-encoded file. The data table is saved from the display specified in <spec>.

See MMEM:STOR for more information.

MMEM:STOR:LIMit command

Overlapped: yes only for EXT:
Delayed result: no
Pass control required: yes only for EXT:
Power-up state: not applicable

Example Statements:

```
OUTPUT 711;"MMEM:STOR:LIM5 'INT:limit_4'"
OUTPUT 711;"Mmemory:Store:Limit6 ""EXT:NEWTABLE"""
```

Command Syntax: MMEMory:STORe:LIMit<spec><sp>{'|"}[<msi>]<filename>{'|"}

<spec>::=a single integer from 1 to 8

<msi>::=EXT:|INT:|RAM:

<filename>::=1 to 10 printable ASCII characters

Description:

Use this command to save a limit table into a binary-encoded file. The data table saved is the one specified in <spec>.

See MMEM:STOR for more information.

MMEM:STOR:MATH command

Overlapped: yes only for EXT:
Delayed result: no
Pass control required: yes only for EXT:
Power-up state: not applicable

Example Statements:

```
OUTPUT 711;"mmem:stor:math ""RAM:Constant8"""
OUTPUT 711;"MMEMORY:STORE:MATH ""EXT:mathfunc"""
```

Command Syntax:

MMEMory:STORe:MATH<sp>{'|"}[<msi>]<filename>{'|"}

<msi>::=EXT:|INT:|RAM:

<filename>::=1 to 10 printable ASCII characters

Description:

Use this command to save math definitions into a binary-encoded file.

The analyzer allows you to define five math functions and five constants. When you save or recall a math file, you are saving or recalling all five function and constant definitions at once. You can not save or recall individual functions or constants.

See MMEM:STOR for more information.

MMEM:STOR:STATe command

Overlapped: yes only for EXT:
Delayed result: no
Pass control required: yes only for EXT:
Power-up state: not applicable

Example Statements:

```
OUTPUT 711;"MMEM:STOR:STAT 'EXT:State_14'"
OUTPUT 711;"MMEMORY:STORE:STATE ""INT:file_state"""
```

Command Syntax: MMEMory:STORe:STATe<sp>{'|"}[<msi>]<filename>{'|"}

<msi>::=EXT:|INT:|RAM:

<filename>::=1 to 10 printable ASCII characters

Description:

Use this command to save the current instrument state (setup) into a binary-encoded file.

NOTE When you save an analyzer setup, you are also saving the following items: all eight limit tables, both data tables, and all five math functions and constants. To decrease the size of an instrument-state file, you can save limit and data tables separately and then clear them before saving the setup.

Some of your measurements may require analyzer setups that are significantly different from the preset state. If you make these measurements often, you may want to save the special setups in instrument-state files. When you change from one setup to another, you can save time by recalling an instrument state rather than sending many individual commands.

See MMEM:STOR for more information.

MMEM:STOR:TRACe command

Overlapped: yes only for EXT:
Delayed result: no
Pass control required: yes only for EXT:
Power-up state: not applicable

Example Statements:

```
OUTPUT 711;"Mmem:Stor:Trac ""RAM:LOG_TRACE"""
OUTPUT 711;"mmemory:store:trace 'INT:Trace_File'"
```

Command Syntax: MMEMory:STORe:TRACe[<spec>]<sp>{'|"}[<msi>]<filename>{'|"}

<spec>::=~:A~|:B|1|2

<msi>::=EXT:|INT:|RAM:

<filename>::=1 to 10 printable ASCII characters

Description:

Use this command to save a trace into a binary-encoded file. The trace will be saved from the display specified in <spec>.

See MMEM:STOR for more information.

PLOTter subsystem

Description:

Commands in this subsystem are used to define plotting parameters and to plot different portions of the analyzer's screen.

PLOT:ADDRess[?] command/query

Overlapped: no
Delayed result: no
Pass control required: no
Power-up state: saved in nonvolatile memory

Example Statements:

```
OUTPUT 711;"Plot:Addr 3"
OUTPUT 711;"Plotter:Address 16"
OUTPUT 711;"plot:addr?"
```

Command Syntax: PLOTter:ADDRess<sp><value>

<value>::=any integer x, where 0 ≤ x ≤ 30 (NRf format)

Query Syntax: PLOTter:ADDRess?

Returned Format: <value><LF><^END>

<value>::=an integer (NR1 format)

Description:

Use this command to enter the address of a plotter connected to the analyzer's HP-IB. What you enter here must match the address setting of the plotter's HP-IB address switches. (Refer to the plotter's documentation for the location of its address switches.)

You can either use numbers or one of two nonnumeric parameters to set the value of PLOT:ADDR. The nonnumeric parameters are:

- UP – increases the current value of PLOT:ADDR by one
- DOWN – decreases the current value of PLOT:ADDR by one

The address last specified with this command is saved in nonvolatile memory when you send the SYST:SAVE command. This means that when you turn the analyzer off and then back on, the address does not change.

The query returns the HP-IB address at which the analyzer expects to find a plotter.

PLOT:DUMP selector

Description:

This command only selects the PLOT:DUMP subsystem. Sending PLOT:DUMP alone does nothing.

PLOT:DUMP:MARKer command

Overlapped: yes
Delayed result: no
Pass control required: yes
Power-up state: not applicable

Example Statements:
```
OUTPUT 711;"PLOT:DUMP:MARK"
OUTPUT 711;"Plotter:Dump:Marker"
```

Command Syntax: PLOTter:DUMP:MARKer

Description:

Use this command to plot the position of active trace's main marker. The SCR:ACT query tells you which track is currently active. The x and y values of the marker are plotted in close proximity to the marker itself.

This command only plots a main marker when SCR:CONT is set to TRAC. Also, it only plots the main marker of the active trace. The SCR:ACT command allows you to select the active trace.

When this command is executed, the active controller on the HP-IB must temporarily pass control to the analyzer. This allows the analyzer to directly control the plot operation. When the operation is completed, the analyzer passes control back. For more information on passing control, see Chapter 2, "Behavior in an HP-IB System."

PLOT:DUMP:SCReen command

Overlapped: yes
Delayed result: no
Pass control required: yes
Power-up state: not applicable

Example Statements:
```
OUTPUT 711;"plot:dump:scr"
OUTPUT 711;"PLOTTER:DUMP:SCREEN"
```

Command Syntax: PLOTter:DUMP:SCReen

Description:

Use this command to plot everything on the analyzer's screen except the softkey labels.

When this command is executed, the active controller on the HP-IB must temporarily pass control to the analyzer. This allows the analyzer to directly control the plot operation. When the operation is completed, the analyzer passes control back. For more information on passing control, see Chapter 2, "Behavior in an HP-IB System."

PLOT:DUMP:TRACe command

Overlapped: yes
Delayed result: no
Pass control required: yes
Power-up state: not applicable

Example Statements:
```
OUTPUT 711;"PLOT:DUMP:TRAC"
OUTPUT 711;"PLOTTER:DUMP:TRACE"
```

Command Syntax: PLOTter:DUMP:TRACe

Description:

Use this command to plot the active trace. The SCR:ACT query tells you which trace is currently active. Only the trace itself is plotted.

This command only plots a trace when SCR:CONT is set to TRAC. Also, it only plots the active trace. The SCR:ACT command allows you to select the active trace.

When this command is executed, the active controller on the HP-IB must temporarily pass control to the analyzer. This allows the analyzer to directly control the plot operation. When the operation is completed, the analyzer passes control back. For more information on passing control, see Chapter 2, "Behavior in an HP-IB System."

PLOT:LTYPe selector

Description:

This command only selects the PLOT:LTYP subsystem. Sending PLOT:LTYP alone does nothing.

PLOT:LTYP:TRACe[?] command/query

Overlapped: no
Delayed result: no
Pass control required: no
Power-up state: −4096

Example Statements:

```
OUTPUT 711;"Plot:Ltyp:Trac 1"
OUTPUT 711;"plotter:ltype:trace:b 6"
OUTPUT 711;"PLOT:LTYP:TRAC2?"
```

Command Syntax: PLOTter:LTYPe:TRACe[<spec>]<sp><value>

<spec>::=~:A~|:B|1|2

<value>::=any integer x, where x = −4096, or 0 ≤ x ≤ 6 (NRf format)

Query Syntax: PLOTter:LTYPe:TRACe[<spec>]?

Returned Format: <value><LF><^END>

<value>::=an integer (NR1 format)

Description:

This command lets you select the line type that will be used to plot the specified trace.

The number you enter is encoded. The meanings of three numbers you can enter are as follows:

- −4096 – solid.
- 1 – dotted.
- 2 – dashed.

You must refer to your plotter documentation for the meanings of 0 and 3-6.

You can either use numbers or one of two nonnumeric parameters to set the value of PLOT:LTYP. The nonnumeric parameters are:

- UP – increases the current value of PLOT:LTYP by one
- DOWN – decreases the current value of PLOT:LTYP by one

The query response indicates which line type is currently selected for the specified trace.

PLOT:PEN selector

Description:

This command only selects the PLOT:PEN subsystem. Sending PLOT:PEN alone does nothing.

PLOT:PEN:ALPHa[?] command/query

Overlapped: no
Delayed result: no
Pass control required: no
Power-up state: 1

Example Statements:

```
OUTPUT 711;"Plot:Pen:Alph 1"
OUTPUT 711;"Plotter:Pen:Alpha 27"
OUTPUT 711;"plot:pen:alph?"
```

Command Syntax: PLOTter:PEN:ALPHa<sp><pen_number>

<pen_number>::=any integer x, where 0 ≤ x ≤ 64 (NRf format)

Query Syntax: PLOTter:PEN:ALPHa?

Returned Format: <value><LF><^END>

<value>::=an integer (NR1 format)

Description:

This command allows you to select the pen that will be used to plot the screen's alphanumeric characters. The number you send specifies one of the pens in the plotter's drafting pen carrousel.

You can either use numbers or one of two nonnumeric parameters to set the value of PLOT:PEN:ALPH. The nonnumeric parameters are:

- UP – increases the current value of PLOT:PEN:ALPH by one
- DOWN – decreases the current value of PLOT:PEN:ALPH by one

The query returns the pen number currently designated to plot the screen's alpha numeric characters.

PLOT:PEN:GRID[?] command/query

Overlapped: no
Delayed result: no
Pass control required: no
Power-up state: 2

Example Statements:

```
OUTPUT 711;"PLOT:PEN:GRID 10"
OUTPUT 711;"Plotter:Pen:Grid 17"
OUTPUT 711;"Plot:Pen:Grid?"
```

Command Syntax: PLOTter:PEN:GRID<sp><pen_number>

<pen_number>::=any integer x, where 0 ≤ x ≤ 64 (NRf format)

Query Syntax: PLOTter:PEN:GRID?

Returned Format: <value><LF><^END>

<value>::=an integer (NR1 format)

Description:

This command allows you to select the pen that will be used to plot the trace grids. The number you send specifies one of the pens in the plotter's drafting pen carrousel.

You can either use numbers or one of two nonnumeric parameters to set the value of PLOT:PEN:GRID. The nonnumeric parameters are:

- UP – increases the current value of PLOT:PEN:GRID by one
- DOWN – decreases the current value of PLOT:PEN:GRID by one

The query returns the pen number currently designated to plot the trace grids.

PLOT:PEN:INITialize — command

Overlapped: no
Delayed result: no
Pass control required: no
Power-up state: not applicable

Example Statements:

```
OUTPUT 711;"plot:pen:init"
OUTPUT 711;"PLOTTER:PEN:INITIALIZE"
```

Command Syntax: PLOTter:PEN:INITialize

Description:

This command initializes all of the plotter pen and line-type parameters to their power-up states. The initialized parameters are:

- PLOT:PEN:ALPH
- PLOT:PEN:GRID
- PLOT:PEN:TRAC
- PLOT:LTYP:TRAC

PLOT:PEN:TRACe[?] command/query

Overlapped: no
Delayed result: no
Pass control required: no
Power-up state: 3 (trace A)
4 (trace B)

Example Statements:

```
OUTPUT 711;"PLOT:PEN:TRAC 1"
OUTPUT 711;"PLOTTER:PEN:TRACE:B 12"
OUTPUT 711;"Plot:Pen:Trac?"
```

Command Syntax: PLOTter:PEN:TRACe[<spec>]<sp><pen_number>

<spec>::=~:A~|:B|1|2

<pen_number>::=any integer x, where 0 ≤ x ≤ 64 (NRf format)

Query Syntax: PLOTter:PEN:TRACe[<spec>]?

Returned Format: <value><LF><^END>

<value>::=an integer (NR1 format)

Description:

This command allows you to select the pen that will be used to plot trace A or trace B. The number you send specifies one of the pens in the plotter's drafting pen carrousel.

You can either use numbers or one of two nonnumeric parameters to set the value of PLOT:PEN:TRAC. The nonnumeric parameters are:

- UP – increases the current value of PLOT:PEN:TRAC by one
- DOWN – decreases the current value of PLOT:PEN:TRAC by one

The query returns the pen number currently designated to plot the specified trace.

PLOT:SPEed[?] command/query

Overlapped: no
Delayed result: no
Pass control required: no
Power-up state: 36

Example Statements:

```
OUTPUT 711;"Plot:Spe 5"
OUTPUT 711;"plotter:speed 36"
OUTPUT 711;"PLOT:SPE?"
```

Command Syntax: PLOTter:SPEed<sp><value>

<value>::=any integer x, where 1 ≤ x ≤ 100 (NRf format)

Query Syntax: PLOTter:SPEed?

Returned Format: <value><LF><^END>

<value>::=an integer (NR1 format)

Description:

This command allows you to specify the plotting speed. The assumed unit for the value entered is cm/s.

You can either use numbers or one of two nonnumeric parameters to set the value of PLOT:SPE. The nonnumeric parameters are:

- UP – increases the current value of PLOT:SPE by one
- DOWN – decreases the current value of PLOT:SPE by one

The query returns the current plotting speed.

PRINter subsystem

Description:

Commands in this subsystem are used to specify a printer address and to print different portions of the analyzer's screen.

PRIN:ADDRess[?] command/query

Overlapped: no
Delayed result: no
Pass control required: no
Power-up state: saved in nonvolatile memory

Example Statements:

```
OUTPUT 711;"Prin:Addr 1"
OUTPUT 711;"Printer:Address 24"
OUTPUT 711;"prin:addr?"
```

Command Syntax: PRINter:ADDRess<sp><value>

<value>::=any integer x, where $0 \leq x \leq 30$ (NRf format)

Query Syntax: PRINter:ADDRess?

Returned Format: <value><LF><^END>

<value>::=an integer (NR1 format)

Description:

Use this command to enter the address of a printer connected to the analyzer's HP-IB. What you enter here must match the address setting of the printer's HP-IB address switches. (Refer to the printer's documentation for the location of its address switches.)

You can either use numbers or one of two nonnumeric parameters to set the value of PRIN:ADDR. The nonnumeric parameters are:

- UP – increases the current value of PRIN:ADDR by one
- DOWN – decreases the current value of PRIN:ADDR by one

The address last specified with this command is saved in nonvolatile memory when you send the SYST:SAVE command. This means that when you turn the analyzer off and then back on, the address does not change.

The query returns the HP-IB address at which the analyzer expects to find a printer.

PRIN:DUMP selector

Description:

This command only selects the PRIN:DUMP subsystem. Sending PRIN:DUMP alone does nothing.

PRIN:DUMP:ALPHa command

Overlapped: yes
Delayed result: no
Pass control required: yes
Power-up state: not applicable

Example Statements:

```
OUTPUT 711;"PRIN:DUMP:ALPH"
OUTPUT 711;"Printer:Dump:Alpha"
```

Command Syntax: PRINter:DUMP:ALPHa

Description:

This command allows you to print any ASCII text currently displayed on the analyzer's screen (except for softkey labels). Trace graphics are not printed.

Text is printed for any setting of SCR:CONT.

PRIN:DUMP:SCReen command

Overlapped: yes
Delayed result: no
Pass control required: yes
Power-up state: not applicable

Example Statements:

```
OUTPUT 711;"prin:dump:scr"
OUTPUT 711;"PRINTER:DUMP:SCREEN"
```

Command Syntax: PRINter:DUMP:SCReen

Description:

Use this command to print everything on the analyzer's screen except the softkey labels.

When this command is executed, the active controller on the HP-IB must temporarily pass control to the analyzer. This allows the analyzer to directly control the print operation. When the operation is completed, the analyzer passes control back. For more information on passing control, see Chapter 2, "Behavior in an HP-IB System."

SCReen subsystem

Description:

Commands in this subsystem are used to control the contents of the analyzer's screen. Some of the commands can be useful for controlling the amount of information plotted or printed with the PLOT:DUMP:SCR or PRIN:DUMP:SCR command.

SCR:ACTive[?] command/query

Overlapped: no
Delayed result: no
Pass control required: no
Power-up state: A

Example Statements:

```
OUTPUT 711;"SCR:ACT A"
OUTPUT 711;"SCREEN:ACTIVE B"
OUTPUT 711;"Scr:Act?"
```

Command Syntax: SCReen:ACTive<sp>{A|B}

Query Syntax: SCReen:ACTive?

Returned Format: {A|B}<LF><^END>

Description:

This command selects the active trace.

Two commands in the PLOTter subsystem, PLOT:DUMP:MARK and PLOT:DUMP:TRAC, act on the active trace. This command allows you to specify which trace will be acted on by those commands.

The query returns A or B, depending on which of the two traces is active.

SCR:ANNotation[?] command/query

Overlapped: no
Delayed result: no
Pass control required: no
Power-up state: 1

Example Statements:

```
OUTPUT 711;"Scr:Ann Off"
OUTPUT 711;"screen:annotation 1"
OUTPUT 711;"SCR:ANN?"
```

Command Syntax: SCReen:ANNotation<sp>{OFF|ON|0|1}

Query Syntax: SCReen:ANNotation?

Returned Format: {0|1}<LF><^END>

Description:

Use this command to enable and disable display of the marker readouts and the x-axis labels.

The query returns 0 if the readouts and labels are blanked, 1 if they are being displayed.

SCR:CONTents[?] command/query

Overlapped: yes only for DCAT, and then only when MMEM:MSI is EXT:
Delayed result: no
Pass control required: yes only for DCAT, and then only when MMEM:MSI is EXT:
Power-up state: TRAC

Example Statements:

```
OUTPUT 711;"Scr:Cont Dcat"
OUTPUT 711;"Screen:Contents Tlog"
OUTPUT 711;"scr:cont?"
```

Command Syntax: SCReen:CONTents<sp><option>

<option>::=APPLication|DCATalog|FLOG|MEMory|STATe|TLOG|TRACe

Query Syntax: SCReen:CONTents?

Returned Format: {APPL|DCAT|FLOG|MEM|STAT|TLOG|TRAC}<LF><^END>

Description:

This command lets you specify the contents of the analyzer's screen. The options are:

- APPLication – displays a list of all applications that are currently loaded into the analyzer
- DCATalog – displays the catalog of the current mass storage device
- FLOG – displays the fault log
- MEMory – displays a summary of memory utilization
- STATe – displays a summary of the analyzer setup
- TLOG – displays the test log
- TRACe – displays one or both of the traces, depending on the setting of SCR:FORM

If you send SCR:CONT DCAT while MMEM:MSI is "EXT:", the active controller on the HP-IB must temporarily pass control to the analyzer. When execution of the command has been completed, the analyzer passes control back. For more information on passing control, see Chapter 2, "Behavior in an HP-IB System."

The query tells you which of the options is currently being displayed.

SCR:FORMat[?] command/query

Overlapped: no
Delayed result: no
Pass control required: no
Power-up state: SING

Example Statements:

```
OUTPUT 711;"SCR:FORM FBAC"
OUTPUT 711;"Screen:Format Ulower"
OUTPUT 711;"Scr:Form?"
```

Command Syntax: SCReen:FORMat<sp><option>

<option>::=FBACk|SINGle|ULOWer

Query Syntax: SCReen:FORMat?

Returned Format: {FBAC|SING|ULOW}<LF><^END>

Description:

This command lets you specify the format for trace displays. It only affects the analyzer's screen when SCR:CONT is set to TRAC. The options are:

- FBACk (front/back) – Two full-height traces are displayed on top of each other. The inactive trace is de-emphasized.
- SINGle – Only the active trace is displayed.
- ULOWer (upper/lower) – Two half high traces are displayed.

The query tells you which of the options is currently selected.

SCR[:STATe][?] command/query

Overlapped: no
Delayed result: no
Pass control required: no
Power-up state: 1

Example Statements:

```
OUTPUT 711;"scr on"
OUTPUT 711;"SCREEN:STATE 0"
OUTPUT 711;"SCR?"
```

Command Syntax: SCReen[:STATe]<sp>{OFF|ON|0|1}

Query Syntax: SCReen[:STATe]?

Returned Format: {0|1}<LF><^END>

Description:

Use this command to enable and disable display of everything on the analyzer's screen except the softkey labels.

In addition to the softkey labels, one other thing appears on the screen when SCR:STAT is OFF—a message indicating that the display has been blanked.

The query returns 0 if the display is blanked, 1 if it is not.

SERVice subsystem

Description:

All commands in this subsystem are used invoke service tests or service adjustment routines. Since these tests and routines should be used only by qualified service personnel, the commands are not described here. See the HP 35660A Service Manual for descriptions.

SOURce subsystem

Description:

Commands in this subsystem are used to define the analyzer's source output.

SOUR:AMPLitude[?] command/query

Description:

SOUR:AMPL is functionally equivalent to SOUR:AMPL:LEV. See the latter command for more details.

SOUR:AMPL[:LEVel][?] command/query

Overlapped: no
Delayed result: yes
Pass control required: no
Power-up state: 0

Example Statements:

```
OUTPUT 711;"SOUR:AMPL 5"
OUTPUT 711;"SOURCE:AMPLITUDE:LEVEL -40DBVRMS"
OUTPUT 711;"Sour:Ampl?"
```

Command Syntax: SOURce:AMPLitude[:LEVel]<sp><value>[<unit>]

<value>::=a decimal number (NRf format)

<unit>::=~V~ |VRMS|DBVPK|DBVRMS

Query Syntax: SOURce:AMPLitude[:LEVel]?

Returned Format: <value><LF><^END>

Description:

Use this command to specify the source output level.

You can either use numbers or one of three nonnumeric parameters to set the value of SOUR:AMPL:LEV. The nonnumeric parameters are:

- UP – increases the current value of SOUR:AMPL:LEV to the next largest allowable value
- DOWN – decreases the current value of SOUR:AMPL:LEV to the next smallest allowable value
- (MARK[:A|:B]:VAL) – sets SOUR:AMPL:LEV to the amplitude of the main marker, even when the marker reference is enabled

The query returns the current source output level in the units last entered. Units are not returned with the value.

SOUR:FREQuency[?] command/query

Description:

SOUR:FREQ is functionally equivalent to SOUR:FREQ:CW. See the latter command for more details.

SOUR:FREQ[:CW][?] command/query

Overlapped: no
Delayed result: yes
Pass control required: no
Power-up state: 10240

Example Statements:

```
OUTPUT 711;"Sour:Freq 100"
OUTPUT 711;"source:frequency:cw 45khz"
OUTPUT 711;"SOUR:FREQ?"
```

Command Syntax: SOURce:FREQuency[:CW]<sp><value>[<unit>]

<value>::=any x, where 0 ≤ x ≤ 115,000 (when units are HZ)

Values should be sent using the NRf format.

<unit>::=~HZ~|KHZ

Query Syntax: SOURce:FREQuency[:CW]?

Returned Format: <value><LF><^END>

Description:

Use this command to specify the frequency of a fixed sine (continuous wave) source output.

You can either use numbers or one of three nonnumeric parameters to set the value of SOUR:FREQ:CW. If you use numbers, the source frequency can be set in 1/64 hertz increments. The nonnumeric parameters are:

- UP – increases the current value of SOUR:FREQ:CW by the increment between points on the x-axis
- DOWN – decreases the current value of SOUR:FREQ:CW by the increment between points on the x-axis
- (MARK[:A|:B]:VAL) – sets SOUR:FREQ:CW to the frequency of the main marker, even when the marker reference is enabled

The query returns the frequency currently specified for the fixed sine source output. The frequency is returned in Hz.

SOUR:FREQ:MODE[?] command/query

Overlapped: no
Delayed result: yes
Pass control required: no
Power-up state: CW

Example Statements:

```
OUTPUT 711;"Sour:Freq:Mode CW"
OUTPUT 711;"Source:Frequency:Mode Pchirp"
OUTPUT 711;"sour:freq:mode?"
```

Command Syntax: SOURce:FREQuency:MODE<sp>{CW|PCHirp|RANDom}

Query Syntax: SOURce:FREQuency:MODE?

Returned Format: {CW|PCH|RAND}<LF><^END>

Description:

Use this command to change the source output mode. The options are:

- CW – Continuous sine wave mode. You can specify the frequency of the sine wave using the SOUR:FREQ:CW command.
- PCHirp – Periodic chirp mode. The periodic chirp waveform is a fast sine sweep across the current frequency span. The sweep repeats with the same period as the current time record.
- RANDom – Random noise mode.

The query indicates which of the three source output modes is selected.

SOUR:STATe[?] command/query

Overlapped: no
Delayed result: yes
Pass control required: no
Power-up state: 0

Example Statements:

```
OUTPUT 711;"SOUR:STAT OFF"
OUTPUT 711;"Source:State 1"
OUTPUT 711;"Sour:Stat?"
```

Command Syntax: SOURce:STATe<sp>{OFF|ON|0|1}

Query Syntax: SOURce:STATe?

Returned Format: {0|1}<LF><^END>

Description:

This command enables and disables the analyzer's source output.

The query returns 0 if source output is disabled, 1 if it is enabled.

STATus subsystem

Description:

Commands in this subsystem give you access to instrument status. They also allow you to specify which status changes will cause the analyzer to request service from the HP-IB controller.

STAT:DEVice selector

Description:

This command only selects the STAT:DEV subsystem. Sending STAT:DEV alone does nothing.

All commands in this subsystem either set or query one of the five registers in the Device Status register set. Decimal weights are assigned to bits in the registers according to the following formula:

$$2^{(bit\_number)}$$

with acceptable values for bit_number being 0 through 15. The value sent with each command or returned with each query is a sum of the decimal weights of all set bits.

Information about the instrument is constantly updated in the Device Status condition register. Two Device Status transition registers then determine how much of that information is reported to the Device Status event register.

The Device Status event register is summarized in bit 7 of the Status Byte register. The Device Status enable register determines how much of the event register information is included in the summary.

For more information on the Device Status register set, see Chapter 5, "Using the HP 35660A's Status Registers."

STAT:DEV:CONDition? query

Overlapped: no
Delayed result: no
Pass control required: no
Power-up state: 128

Example Statement: `OUTPUT 711;"STAT:DEV:COND?"`

Query Syntax: STATus:DEVice:CONDition?

Returned Format: <value><LF><^END>

<value>::=an integer (NR1 format)

Description:

This query returns the current state of the Device Status condition register. The state is returned as a sum of the decimal weights of all set bits. (See STAT:DEV for formula.)

Each bit in the register reports on a particular instrument condition. A bit is set to 1 when the condition is true and reset to 0 when the condition is false. For information on the conditions assigned to each bit, see Chapter 5, "Using the HP 35660A's Status Registers."

This query does not change the state of any bits in the Device Status condition register.

For more information, see STAT:DEV.

STAT:DEV:ENABle[?] command/query

Overlapped: no
Delayed result: no
Pass control required: no
Power-up state: 0

Example Statements:

```
OUTPUT 711;"Stat:Dev:Enab 16"
OUTPUT 711;"status:device:enable 224"
OUTPUT 711;"STAT:DEV:ENAB?"
```

Command Syntax: STATus:DEVice:ENABle<sp><value>

<value>::=any integer x, where 0 ≤ x ≤ 65,535 (NRf format)

Query Syntax: STATus:DEVice:ENABle?

Returned Format: <value><LF><^END>

<value>::=an integer (NR1 format)

Description:

This command allows you to set bits in the Device Status enable register. Send the sum of the decimal weights of all bits you want to set. (See STAT:DEV for formula.)

When an enable register bit is set to 1, the corresponding bit of the Device Status event register is enabled. The enabled bit will be included in the Device Status summary.

The Device Status summary is reported to bit 7 of the Status Byte register. Bit 7 is only set if both of the following are true:

- One or more bits of the Device Status event register are set
- At least one of the set bits is enabled by a corresponding bit in the Device Status enable register

All bits in the Device Status enable register are initialized to 0 when the instrument is turned on. However, the current setting of bits is not modified when you send the *RST command.

The query returns the current state of the Device Status enable register The state is returned as a sum of the decimal weight of all set bits.

For more information, see STAT:DEV.

STAT:DEV:EVENt? query

Overlapped: no
Delayed result: no
Pass control required: no
Power-up state: 0

Example Statement: `OUTPUT 711;"Stat:Dev:Even?"`

Query Syntax: STATus:DEVice:EVENt?

Returned Format: <value><LF><^END>

<value>::=an integer (NR1 format)

Description:

This query returns the current state of the Device Status event register. The state is returned as a sum of the decimal weights of all set bits. (See STAT:DEV for formula.) The register is cleared after being read by this query.

Each bit in the Device Status event register is set to 1 when the corresponding bit of the Device Status condition register makes a transition from 0 to 1 or from 1 to 0. However, this is only true if the transition is enabled by one of the two Device Status transition registers.

Once set, a bit in the Device Status event register remains set. It disregards any further changes in the corresponding condition register bit. To reset event register bits to 0, you must either query the event register or send the *CLS command.

For more information, see STAT:DEV.

STAT:DEV:NTR[?] command/query

Overlapped: no
Delayed result: no
Pass control required: no
Power-up state: 0

Example Statements:

```
OUTPUT 711;"STAT:DEV:NTR 4"
OUTPUT 711;"Status:Device:Ntr 13"
OUTPUT 711;"Stat:Dev:Ntr?"
```

Command Syntax: STATus:DEVice:NTR<sp><value>

<value>::=any integer x, where 0 ≤ x ≤ 65,535 (NRf format)

Query Syntax: STATus:DEVice:NTR?

Returned Format: <value><LF><^END>

<value>::=an integer (NR1 format)

Description:

This command allows you to set bits in the Device Status negative transition register (NTR). Send the sum of the decimal weights of all bits you want to set. (See STAT:DEV for formula.)

Bits in this register are used to enable corresponding bits in the Device Status condition register. Each bit in the condition register indicates whether a particular instrument condition is true or false. When a bit in the condition register makes a transition from true to false, that transition is only reported to the event register if the corresponding NTR bit is set to 1.

All bits in the Device Status NTR are initialized to 0 when the instrument is turned on. However, the current setting of bits is not modified when you send the *RST command.

The query returns the current state of the Device Status NTR. The state is returned as a sum of the decimal weight of all set bits.

For more information, see STAT:DEV.

STAT:DEV:PTR[?] command/query

Overlapped: no
Delayed result: no
Pass control required: no
Power-up state: 0

Example Statements:
```
OUTPUT 711;"stat:dev:ptr 2"
OUTPUT 711;"STATUS:DEVICE:PTR 7"
OUTPUT 711;"STAT:DEV:PTR?"
```

Command Syntax: STATus:DEVice:PTR<sp><value>

<value>::=any integer x, where 0 ≤ x ≤ 65,535 (NRf format)

Query Syntax: STATus:DEVice:PTR?

Returned Format: <value><LF><^END>

<value>::=an integer (NR1 format)

Description:

This command allows you to set bits in the Device Status positive transition register (PTR). Send the sum of the decimal weights of all bits you want to set. (See STAT:DEV for formula.)

Bits in this register are used to enable corresponding bits in the Device Status condition register. Each bit in the condition register indicates whether a particular instrument condition is true or false. When a bit in the condition register makes a transition from false to true, that transition is only reported to the event register if the corresponding PTR bit is set to 1.

All bits in the Device Status PTR are initialized to 0 when the instrument is turned on. However, the current setting of bits is not modified when you send the *RST command.

The query returns the current state of the Device Status PTR. The state is returned as a sum of the decimal weight of all set bits.

For more information, see STAT:DEV.

STAT:DINTegrity selector

Description:

This command only selects the STAT:DINT subsystem. Sending STAT:DINT alone does nothing.

All commands in this subsystem either set or query one of the five registers in the Data Integrity register set. Decimal weights are assigned to bits in the registers according to the following formula:

$$2^{(bit\_number)}$$

with acceptable values for bit_number being 0 through 15. The value sent with each command or returned with each query is a sum of the decimal weights of all set bits.

Information that affects the validity of measurement results is constantly updated in the Data Integrity condition register. Two Data Integrity transition registers then determine how much of that information is reported to the Data Integrity event register.

The Data Integrity event register is summarized in bit 4 of the Device Status condition register. The Data Integrity enable register determines how much of the event register information is included in the summary.

For more information on the Data Integrity register set, see Chapter 5, "Using the HP 35660A's Status Registers."

STAT:DINT:CONDition? query

Overlapped: no
Delayed result: no
Pass control required: no
Power-up state: 0

Example Statement: `OUTPUT 711;"STAT:DINT:COND?"`

Query Syntax: STATus:DINTegrity:CONDition?

Returned Format: <value><LF><^END>

<value>::=an integer (NR1 format)

Description:

This query returns the current state of the Data Integrity condition register. The state is returned as a sum of the decimal weights of all set bits. (See STAT:DINT for formula.)

Each bit in the register reports on a particular instrument condition. A bit is set to 1 when the condition is true and reset to 0 when the condition is false. For information on the conditions assigned to each bit, see Chapter 5, "Using the HP 35660A's Status Registers."

This query does not change the state of any bits in the Data Integrity condition register.

For more information, see STAT:DINT.

STAT:DINT:ENABle[?] command/query

Overlapped: no
Delayed result: no
Pass control required: no
Power-up state: 0

Example Statements:
```
OUTPUT 711;"Stat:Dint:Enab 2"
OUTPUT 711;"status:dintegrity:enable 768"
OUTPUT 711;"STAT:DINT:ENAB?"
```

Command Syntax: STATus:DINTegrity:ENABle<sp><value>

<value>::=any integer x, where 0 ≤ x ≤ 65,535 (NRf format)

Query Syntax: STATus:DINTegrity:ENABle?

Returned Format: <value><LF><^END>

<value>::=an integer (NR1 format)

Description:

This command allows you to set bits in the Data Integrity enable register. Send the sum of the decimal weights of all bits you want to set. (See STAT:DINT for formula.)

When an enable register bit is set to 1, the corresponding bit of the Data Integrity event register is enabled. The enabled bit will be included in the Data Integrity summary.

The Data Integrity summary is reported to bit 4 of the Device Status condition register. Bit 4 is only set if both of the following are true:

- One or more bits of the Data Integrity event register are set
- At least one of the set bits is enabled by a corresponding bit in the Data Integrity enable register

All bits in the Data Integrity enable register are initialized to 0 when the instrument is turned on. However, the current setting of bits is not modified when you send the *RST command.

The query returns the current state of the Data Integrity enable register The state is returned as a sum of the decimal weight of all set bits.

For more information, see STAT:DINT.

STAT:DINT:EVENt? query

Overlapped: no
Delayed result: no
Pass control required: no
Power-up state: 0

Example Statement: `OUTPUT 711;"Stat:Dint:Even?"`

Query Syntax: STATus:DINTegrity:EVENt?

Returned Format: <value><LF><^END>

<value>::=an integer (NR1 format)

Description:

This query returns the current state of the Data Integrity event register. The state is returned as a sum of the decimal weights of all set bits. (See STAT:DINT for formula.) The register is cleared after being read by this query.

Each bit in the Data Integrity event register is set to 1 when the corresponding bit of the Data Integrity condition register makes a transition from 0 to 1 or from 1 to 0. However, this is only true if the transition is enabled by one of the two Data Integrity transition registers.

Once set, a bit in the Data Integrity event register remains set. It disregards any further changes in the corresponding condition register bit. To reset event register bits to 0, you must either query the event register or send the *CLS command.

For more information, see STAT:DINT.

STAT:DINT:NTR[?] command/query

Overlapped: no
Delayed result: no
Pass control required: no
Power-up state: 0

Example Statements:

```
OUTPUT 711;"STAT:DINT:NTR 16"
OUTPUT 711;"Status:Dintegrity:Ntr 12"
OUTPUT 711;"Stat:Dint:Ntr?"
```

Command Syntax: STATus:DINTegrity:NTR<sp><value>

<value>::=any integer x, where 0 ≤ x ≤ 65,535 (NRf format)

Query Syntax: STATus:DINTegrity:NTR?

Returned Format: <value><LF><^END>

<value>::=an integer (NR1 format)

Description:

This command allows you to set bits in the Data Integrity negative transition register (NTR). Send the sum of the decimal weights of all bits you want to set. (See STAT:DINT for formula.)

Bits in this register are used to enable corresponding bits in the Data Integrity condition register. Each bit in the condition register indicates whether a particular instrument condition is true or false. When a bit in the condition register makes a transition from true to false, that transition is only reported to the event register if the corresponding NTR bit is set to 1.

All bits in the Data Integrity NTR are initialized to 0 when the instrument is turned on. However, the current setting of bits is not modified when you send the *RST command.

The query returns the current state of the Data Integrity NTR. The state is returned as a sum of the decimal weight of all set bits.

For more information, see STAT:DINT.

STAT:DINT:PTR[?] command/query

Overlapped: no
Delayed result: no
Pass control required: no
Power-up state: 0

Example Statements:

```
OUTPUT 711;"stat:dint:ptr 1"
OUTPUT 711;"STATUS:DINTEGRITY:PTR 48"
OUTPUT 711;"Stat:Dint:Ptr?"
```

Command Syntax: STATus:DINTegrity:PTR<sp><value>

<value>::=any integer x, where 0 ≤ x ≤ 65,535 (NRf format)

Query Syntax: STATus:DINTegrity:PTR?

Returned Format: <value><LF><^END>

<value>::=an integer (NR1 format)

Description:

This command allows you to set bits in the Data Integrity positive transition register (PTR). Send the sum of the decimal weights of all bits you want to set. (See STAT:DINT for formula.)

Bits in this register are used to enable corresponding bits in the Data Integrity condition register. Each bit in the condition register indicates whether a particular instrument condition is true or false. When a bit in the condition register makes a transition from false to true, that transition is only reported to the event register if the corresponding PTR bit is set to 1.

All bits in the Data Integrity PTR are initialized to 0 when the instrument is turned on. However, the current setting of bits is not modified when you send the *RST command.

The query returns the current state of the Data Integrity PTR. The state is returned as a sum of the decimal weight of all set bits.

For more information, see STAT:DINT.

STAT:USER selector

Description:

This command only selects the STAT:USER subsystem. Sending STAT:USER alone does nothing.

The commands in this subsystem either set or query one of the two registers in the User Status register set. Decimal weights are assigned to bits in the registers according to the following formula:

$$2^{(\text{bit\_number})}$$

with acceptable values for bit_number being 0 through 15. The value sent with each command or returned with each query is a sum of the decimal weights of all set bits.

The User Status event register is summarized in bit 0 of the Status Byte register. The User Status enable register determines how much of the event register information is included in the summary.

For more information on the User Status register set, see Chapter 5, "Using the HP 35660A's Status Registers."

STAT:USER:ENABle[?] command/query

Overlapped: no
Delayed result: no
Pass control required: no
Power-up state: 0

Example Statements:

```
OUTPUT 711;"STAT:USER:ENAB 256"
OUTPUT 711;"STATUS:USER:ENABLE 53"
OUTPUT 711;"Stat:User:Enab?"
```

Command Syntax: STATus:USER:ENABle<sp><value>

<value>::=any integer x, where 0 ≤ x ≤ 65,535 (NRf format)

Query Syntax: STATus:USER:ENABle?

Returned Format: <value><LF><^END>

<value>::=an integer (NR1 format)

Description:

This command allows you to set bits in the User Status enable register. Send the sum of the decimal weights of all bits you want to set. (See STAT:USER for formula.)

When an enable register bit is set to 1, the corresponding bit of the User Status event register is enabled. The enabled bit will be included in the User Status summary.

The User Status summary is reported to bit 0 of the Status Byte register. Bit 0 is only set if both of the following are true:

- One or more bits of the User Status event register are set
- At least one of the set bits is enabled by a corresponding bit in the User Status enable register

All bits in the User Status enable register are initialized to 0 when the instrument is turned on. However, the current setting of bits is not modified when you send the *RST command.

The query returns the current state of the User Status enable register The state is returned as a sum of the decimal weight of all set bits.

For more information, see STAT:USER.

STAT:USER:EVENt? query

Overlapped: no
Delayed result: no
Pass control required: no
Power-up state: 0

Example Statement: `OUTPUT 711;"Stat:User:Even?"`

Query Syntax: STATus:USER:EVENt?

Returned Format: <value><LF><^END>

<value>::=an integer (NR1 format)

Description:

This query returns the current state of the User Status event register. The state is returned as a sum of the decimal weights of all set bits. (See STAT:USER for formula.) The register is cleared after being read by this query.

The analyzer has ten User SRQ softkeys. A bit in the User Status event register is set to 1 when the corresponding User SRQ softkey is pressed. A bit can also be set by sending the appropriate STAT:USER:PULS command via the HP-IB.

Once set, a bit in the User Status event register remains set. To reset event register bits to 0, you must either query the event register or send the *CLS command.

For more information, see STAT:USER.

STAT:USER:PULSe command

Overlapped: no
Delayed result: no
Pass control required: no
Power-up state: not applicable

Example Statements:

```
OUTPUT 711;"Stat:User:Puls 6"
OUTPUT 711;"Status:User:Pulse 288"
```

Command Syntax: STATus:USER:PULSe<sp><value>

<value>::=any integer x, where 0 ≤ x ≤ 65,535 (NRf format)

Description:

This command allows you to set bits in the User Status event register. Send the sum of the decimal weights of all bits you want to set. (See STAT:USER for formula.)

SWEep subsystem

Description:

The single command in this subsystem is used to specify the length of the time record you want to analyze.

SWE:TIME[?] command/query

Overlapped: no
Delayed result: yes
Pass control required: no
Power-up state: 0.003906

Example Statements:

```
OUTPUT 711;"SWE:TIME 2048"
OUTPUT 711;"Sweep:Time 125MS"
OUTPUT 711;"Swe:Time?"
```

Command Syntax: SWEep:TIME<sp><value>[<unit>]

<value>::=any x, where $x=(400\times2^{n})/max\_span$ (NRf format)

n::=an integer from 0 through 19

max_span::=102,400 for one-channel measurements

51,200 for two-channel measurements

<unit>::=~S~|MS|US

Query Syntax: SWEep:TIME?

Returned Format: <value><LF><^END>

<value>::=a decimal number (NRf format)

Description:

Use this command to specify the length of the time record you want to analyze.

When you send this command, two other values are adjusted. FREQ:SPAN is adjusted so the following formula is true:

$$FREQ:SPAN = 400/SWE:TIME$$

FREQ:SPAN is in Hz and SWE:TIME is in seconds.

The other value that is adjusted is either FREQ:CENT or FREQ:STAR, depending on which of the two values was last set. The value last set remains fixed while the other is adjusted to make the following formula true:

$$FREQ:SPAN = (FREQ:CENT - FREQ:STAR) \times 2$$

All values must be in Hz.

You can either use numbers or one of two nonnumeric parameters to set the value of SWE:TIME. If you use numbers, SWE:TIME is set to the closest allowable record length that is greater than or equal to the number sent. The nonnumeric parameters are:

- UP – increases SWE:TIME to the next largest allowable value
- DOWN – decreases SWE:TIME to the next smallest allowable value

The query returns the current record length. The unit for the returned value is seconds.

SYSTem subsystem

Description:

Commands in this subsystem give you access to a number of miscellaneous analyzer functions.

SYST:ADDRess[?] command/query

Overlapped: no
Delayed result: no
Pass control required: no
Power-up state: saved in nonvolatile memory

Example Statements:

```
OUTPUT 711;"syst:addr 11"
OUTPUT 711;"SYSTEM:ADDR 20"
OUTPUT 711;"SYST:ADDR?"
```

Command Syntax: SYSTem:ADDRess<sp><value>

<value>::=an integer from 0 through 30 (NRf format)

Query Syntax: SYSTem:ADDRess?

Returned Format: <value><LF><^END>

<value>::=an integer (NR1 format)

Description:

This command allows you to set the analyzer's HP-IB address. When a controller sends commands to the analyzer via the HP-IB, they must be sent to the address you specify with this command.

NOTE The analyzer's address does not change until SYST:ADDR has been processed by the command parser.

You can either use numbers or one of two nonnumeric parameters to set the value of SYST:ADDR. The nonnumeric parameters are:

- UP – increases the current value of SYST:ADDR by one
- DOWN – decreases the current value of SYST:ADDR by one

NOTE The address you set with this command will be saved to nonvolatile memory only if you issue the SYST:SAVE command.

The query returns the HP-IB address that must be used for sending commands to the analyzer.

SYST:BEEPer[?] command/query

Overlapped: no
Delayed result: no
Pass control required: no
Power-up state: 1

Example Statements:

```
OUTPUT 711;"SYST:BEEP OFF"
OUTPUT 711;"SYSTEM:BEEPER 1"
OUTPUT 711;"Syst:Beep?"
```

Command Syntax: SYSTem:BEEPer<sp>{OFF|ON|0|1}

Query Syntax: SYSTem:BEEPer?

Returned Format: {0|1}<LF><^END>

Description:

This command enables and disables the analyzer's beeper.

The query returns 0 if the beeper is disabled, 1 if it is enabled.

SYST:DATE[?] command/query

Overlapped: no
Delayed result: no
Pass control required: no
Power-up state: saved in nonvolatile memory

Example Statements:

```
OUTPUT 711;"Syst:Date 1988,1,1"
OUTPUT 711;"system:date 1989,07,14"
OUTPUT 711;"SYST:DATE?"
```

Command Syntax: SYSTem:DATE<sp><year>,<month>,<day>

<year>::=an integer from 1987 to 9999

<month>::=an integer from 1 through 12

<day>::=an integer from 1 through 31

Query Syntax: SYSTem:DATE?

Returned Format: <year>,<month>,<day><LF><^END>

All values are returned in NR1 format.

Description:

Use this command to reset the date portion of the analyzer's clock. The date is saved in nonvolatile memory when you send the SYST:SAVE command.

NOTE The clock does not continue to run when the analyzer is turned off. When you turn the analyzer off and back on, the time and date saved in nonvolatile memory are recalled to initialize the clock.

The query returns the current date according to the analyzer's clock.

SYST:DTIMe[?] command/query

Overlapped: no
Delayed result: no
Pass control required: no
Power-up state: saved in nonvolatile memory

Example Statements:

```
OUTPUT 711;"Syst:Dtim 1988,12,24,02,30,45"
OUTPUT 711;"System:Dtime 2001,1,4,14,45,57"
OUTPUT 711;"syst:dtim?"
```

Command Syntax: SYSTem:DTIMe<sp><year>,<month>,<day>,<hour>,<minute>,<second>

<year>::=an integer from 1987 to 9999
<month>::=an integer from 1 through 12
<day>::=an integer from 1 through 31
<hour>::=an integer from 0 through 23
<minute>::=an integer from 0 through 59
<second>::=an integer from 0 through 59

Query Syntax: SYSTem:DTIMe?

Returned Format: <year>,<month>,<day>,<hour>,<minute>,<second><LF><^END>

All values are returned in NR1 format.

Description:

Use this command to reset both the date and time portions of the analyzer's clock. The date and time are saved in nonvolatile memory when you send the SYST:SAVE command.

NOTE The clock does not continue to run when the analyzer is turned off. When you turn the analyzer off and back on, the time and date saved in nonvolatile memory are recalled to initialize the clock.

The query returns the current date and time according to the analyzer's clock.

SYST:DUMP selector

Description:

This command only selects the SYST:DUMP subsystem. Sending SYST:DUMP alone does nothing.

SYST:DUMP:PLOTter[?] command/query

Overlapped: yes
Delayed result: no
Pass control required: yes only for command
Power-up state: not applicable

Example Statements:
```
OUTPUT 711;"SYST:DUMP:PLOT 4"
OUTPUT 711;"System:Dump:Plotter 6"
OUTPUT 711;"Syst:Dump:Plot?"
```

Command Syntax: SYSTem:DUMP:PLOTter<sp>[<value>]

<value>::=an integer from 0 through 30 (NRf format)

Query Syntax: SYSTem:DUMP:PLOTter?

Returned Format: see description

Description:

Use this command to plot everything that appears on the analyzer's screen except the softkey labels.

The value you send with this command specifies the plotter's HP-IB address. If you don't specify an address, the value in PLOT:ADDR will be used.

When this command is executed, the active controller on the HP-IB must temporarily pass control to the analyzer. This allows the analyzer to directly control the plot operation. When the operation is completed, the analyzer must pass control back. For more information on passing control, see Chapter 2, "Behavior in an HP-IB System."

When you send the query, the analyzer dumps its screen contents to the controller using the HP-GL format.

SYST:DUMP:PRINter[?] command/query

Overlapped: yes
Delayed result: no
Pass control required: yes only for command
Power-up state: not applicable

Example Statements:
```
OUTPUT 711;"syst:dump:prin 3"
OUTPUT 711;"SYSTem:DUMP:PRINter 7"
OUTPUT 711;"SYST:DUMP:PRIN?"
```

Command Syntax: SYSTem:DUMP:PRINter<sp>[<value>]

<value>::=an integer from 0 through 30 (NRf format)

Query Syntax: SYSTem:DUMP:PRINter?

Returned Format: see description

Description:

Use this command to print everything that appears on the analyzer's screen except the softkey labels.

The value you send with this command specifies the printer's HP-IB address. If you don't specify an address, the value in PRIN:ADDR will be used.

When this command is executed, the active controller on the HP-IB must temporarily pass control to the analyzer. This allows the analyzer to directly control the print operation. When the operation is completed, the analyzer must pass control back. For more information on passing control, see Chapter 2, "Behavior in an HP-IB System."

When you send the query, the analyzer dumps its screen contents to the controller using the HP raster dump format.

SYST:ERRor? query

Overlapped: no
Delayed result: no
Pass control required: no
Power-up state: 0,""

Example Statement: `OUTPUT 711;"SYST:ERR?"`

Query Syntax: SYSTem:ERRor?

Returned Format: <error_number>,"<error_name>;<message>"<LF><^END>

<error_number>::=an integer (NR1 format)

<error_name>::=name associated with error number (see Appendix D)

<message>::=message explaining why error was generated

The <error_name> and <message> are returned as a series of ASCII characters.

Description:

This query returns the contents of the analyzer's error queue.

One error is returned in response to each query. Errors are returned until the error queue is empty, with the most recent error being returned last. When the queue is empty, the error number will be 0.

The query returns up to 255 characters. If an error exceeds this limit, it is truncated to 255 characters. The limit includes the error number, the quotes, and the characters within the quotes.

SYST:FLOG? query

Overlapped: no
Delayed result: no
Pass control required: no
Power-up state: #0

Example Statement:
```
OUTPUT 711;"Syst:Flog?"
```

Query Syntax: SYSTem:FLOG?

Returned Format: #0{<fault><LF>}...<^END>

<fault>::=<fault_number>,<count>,"<description>"

The length of <fault> never exceeds 83 characters (including quotes and spaces).

<fault_number>::=an integer (NR1 format)

This number specifies the type of fault. It does not represent the fault's order in the fault log.

<count>::=an integer (NR1 format)

<description>::=a description of the fault

The description is returned as a series of ASCII characters.

Description:

This query returns the contents of the analyzer's fault log.

The contents are returned starting with the first line of the fault log and ending with the last. Line-feed characters separate each line.

SYST:FLOG:CLEar command

Overlapped: no
Delayed result: no
Pass control required: no
Power-up state: not applicable

Example Statements:
```
OUTPUT 711;"Syst:Flog:Cle"
OUTPUT 711;"System:Flog:Clear"
```

Command Syntax: SYSTem:FLOG:CLEar

Description:

Use this command to clear the fault log.

SYST:FLOG:ENTRy selector

Description:

This command only selects the SYST:FLOG:ENTR subsystem. Sending SYST:FLOG:ENTR alone does nothing.

SYST:FLOG:ENTR:DESCription command

Overlapped: no
Delayed result: no
Pass control required: no
Power-up state: 0

Example Statements:

```
OUTPUT 711;"Syst:FLOG:ENTR:DESC 10"
OUTPUT 711;"System:Flog:Entry:Description 7"
```

Command Syntax: SYSTem:FLOG:ENTRy:DESCription<sp><line_number>

<line_number>::=any integer x, where x ≥ 1

Description:

This command expands the specified line of the fault log. A more detailed description of the line will be displayed on the analyzer's screen.

The fault log must be displayed before you send this command. Use the SCR:CONT FLOG command to display the fault log.

SYST:HEADer[?] command/query

Overlapped: no
Delayed result: no
Pass control required: no
Power-up state: 0

Example Statements:

```
OUTPUT 711;"syst:head on"
OUTPUT 711;"SYSTEM:HEADER 0"
OUTPUT 711;"SYST:HEAD?"
```

Command Syntax: SYSTem:HEADer<sp>{OFF|ON|0|1}

Query Syntax: SYSTem:HEADer?

Returned Format: {0|1}<LF><^END>

Description:

This command allows you to indicate whether or not query responses should be preceded by a header.

A header essentially echoes the query you sent before returning the requested information. For example, if you send the query Sour:Freq:Mode? while the source output mode is random noise, the possible responses are:

- SOUR:FREQ:MODE<sp>RAND<LF><^END> (with SYST:HEAD ON)
- RAND<LF><^END> (with SYST:HEAD OFF)

The header is just the query you sent, with a space replacing the question mark.

The SYST:HEAD query returns 0 if headers will not be returned. It returns SYST:HEAD 1 if headers will be returned.

SYST:MEMory? query

Overlapped: no
Delayed result: no
Pass control required: no
Power-up state: instrument dependent

Example Statement: `OUTPUT 711;"syst:mem?"`

Query Syntax: SYSTem:MEMory?

Returned Format: <value><LF><^END>

<value>::=an integer (NR1 format)

Description:

The response to this query tells you how much total RAM (random access memory) the analyzer contains. It does not tell you how much memory is available. The value is returned in bytes.

SYST:SAVE command

Overlapped: no
Delayed result: no
Pass control required: no
Power-up state: not applicable

Examples statement: `OUTPUT 711;"system:save"`

Command Syntax: SYSTem:SAVE

Description:

This command allows you save, in nonvolatile memory, the values last sent with selected HP-IB commands. The commands are:

*ESE	MMEM:MSI:VOL
*PCB	PLOT:ADDR
*PSC	PRIN:ADDR
*SRE	SYST:ADDR
MMEM:LOAD:APPL:AUTO	SYST:DATE
MMEM:MSI	SYST:DTIM
MMEM:MSI:ADDR	SYST:TIME
MMEM:MSI:UNIT	

Also saved is a flag that indicates whether the analyzer is currently configured as the HP-IB system controller or as an addressable-only HP-IB device. All of the listed commands have two things in common:

- *RST does not change the values last sent with these commands.
- The values last saved with SYST:SAVE are recalled when you turn the analyzer on.

NOTE The analyzer uses an EEPROM chip for nonvolatile memory. After you have changed the state of an EEPROM about 10,000 times, it may no longer be able retain its settings when you turn the analyzer off. For this reason, it is best if you do not repeatedly send the SYST:SAVE command.

SYST:SERial? query

Overlapped: no
Delayed result: no
Pass control required: no
Power-up state: instrument dependent

Example Statement: `OUTPUT 711;"SYST:SER?"`

Query Syntax: SYSTem:SERial?

Returned Format: <value><LF><^END>

<value>::=10 ASCII characters

Description:

This query returns the serial number of the analyzer.

SYST:SET[?] command/query

Overlapped: no
Delayed result: no
Pass control required: no
Power-up state: variable

Example Statements:

```
OUTPUT 711;"SYST:SET <block_data>"
OUTPUT 711;"SYSTem:SET <block_data>"
OUTPUT 711;"SYST:SET?"
```

Command Syntax: SYSTem:SET<sp><block_data>

<block_data> takes one of two forms, depending on whether you are sending ASCII-encoded or binary-encoded data.

When you are sending ASCII-encoded data (SYST:SET:FORM ASC):

<block_data>::=#0{<line><LF>}...<^END>

<line>::=data on a particular line of the record

<line><LF> is repeated until all lines have been sent.

When you are sending binary-encoded data (SYST:SET:FORM BIN):

<block_data>::=#<byte><length_bytes>{<data_byte>}...

<byte>::=one ASCII-encoded byte that specifies the number of length bytes to follow

<length_bytes>::=ASCII-encoded bytes that specify the number of data bytes to follow

<data_byte>::=one byte of the record's data

<data_byte> is repeated until the number of bytes specified in <length_bytes> has been sent.

Query Syntax: SYSTem:SET?

Returned Format: <block_data>

Description:

Use this command to transfer a complete instrument state from the controller to the analyzer via the HP-IB. The complete instrument state is described in Chapter 4, "Transferring Data."

The query transfers a complete instrument state from the analyzer to the controller.

SYST:SET:FORM[?] command/query

Overlapped: no
Delayed result: no
Pass control required: no
Power-up state: ASC

Examples statements:
```
OUTPUT 711;"Syst:Set:Form Asc"
OUTPUT 711;"system:set:form bin"
OUTPUT 711;"SYST:SET:FORM?"
```

Command Syntax: SYSTem:SET:FORM<sp>{ASCii|BINary}

Query Syntax: SYSTem:SET:FORM?

Returned Format: {ASC|BIN}<LF><^END>

Description:

An instrument state can be ASCII-encoded or binary-encoded when it is transferred between the analyzer and a controller. This command lets you specify the type of encoding to be used for the transfer.

The query returns ASC or BIN, depending on the option specified.

SYST:TIME[?] command/query

Overlapped: no
Delayed result: no
Pass control required: no
Power-up state: saved in nonvolatile memory

Example Statements:

```
OUTPUT 711;"SYST:TIME 07,35,21"
OUTPUT 711;"System:Time 8,17,4"
OUTPUT 711;"Syst:Time?"
```

Command Syntax: SYSTem:TIME<sp><hour>,<minute>,<second>

<hour>::=an integer from 0 through 23
<minute>::=an integer from 0 through 59
<second>::=an integer from 0 through 59

Query Syntax: SYSTem:TIME?

Returned Format: <hour>,<minute>,<second><LF><^END>

All values are returned in NR1 format.

Description:

Use this command to reset the time portion of the analyzer's clock. The time is saved in nonvolatile memory when you send the SYST:SAVE command.

NOTE The clock does not continue to run when the analyzer is turned off. When you turn the analyzer off and back on, the time and date saved in nonvolatile memory are recalled to initialize the clock.

The query returns the current time according to the analyzer's clock.

TEST subsystem

Description:

Most of the commands in this subsystem are used to invoke service tests. Since these tests should be used only by qualified service personnel, the commands are not described here. See the HP 35660A Service Manual for descriptions.

There are four commands in the subsystem that allow you to run the long and short confidence tests and to determine whether the tests passed or failed.

TEST:LCONfidence command

Overlapped: yes
Delayed result: no
Pass control required: no
Power-up state: not applicable

Example Statement: `OUTPUT 711;"TEST:LCON"`

Command Syntax: TEST:LCONfidence

Description:

Use this command to start the long confidence test.

The long confidence test is actually a series of individual tests that check various analyzer functions. Results of the individual tests are reported to the analyzer's test log. The overall result of the long confidence test is available over the HP-IB via the TEST:LCON:RES query.

If the long confidence test fails, contact a qualified service person.

TEST:LCON:RESult? query

Overlapped: no
Delayed result: no
Pass control required: no
Power-up state: 0

Example Statement: `OUTPUT 711;"Test:Lcon:Res?"`

Query Syntax: TEST:LCONfidence:RESult?

Returned Format: {0|1}<LF><^END>

Description:

This query tells you whether or not the analyzer passed the last long confidence test. The query returns 0 if the analyzer failed, 1 if it passed.

If the long confidence test fails, contact a qualified service person.

TEST:RESult? query

Overlapped: no
Delayed result: no
Pass control required: no
Power-up state: 1

Example Statement: `OUTPUT 711;"test:result?"`

Query Syntax: TEST:RESult?

Returned Format: {0|1}<LF><^END>

Description:

This query tells you whether or not the analyzer passed the last self-test. The query returns 0 if the analyzer failed, 1 if it passed.

NOTE The response to this query may not accurately reflect the results of the long confidence test. Use the TEST:LCON:RES query to return the results of that test.

If any self-test fails, contact a qualified service person.

TEST:SHORt command

Overlapped: yes
Delayed result: no
Pass control required: no
Power-up state: not applicable

Example Statement: `OUTPUT 711;"TEST:SHORT"`

Command Syntax: TEST:SHORt

Description:

Use this command to start the short confidence test.

The short confidence test causes the analyzer to self-calibrate, and then compare the calibration results to specified limits. If the results are within the specified limits, the analyzer passes the quick confidence test. The result of the quick confidence test is reported to the analyzer's test log, but is also available over the HP-IB via the TEST:RES query.

If the short confidence test fails, contact a qualified service person.

TRACe subsystem

Description:

This subsystem has two main purposes:

- It allows you to specify which measurement results will appear in each of the two displays
- It provides access to the raw measurement data (data that has not been transformed into the current display coordinates)

The following diagram shows you the difference between data available in the TRAC subsystem and the DISP subsystem:

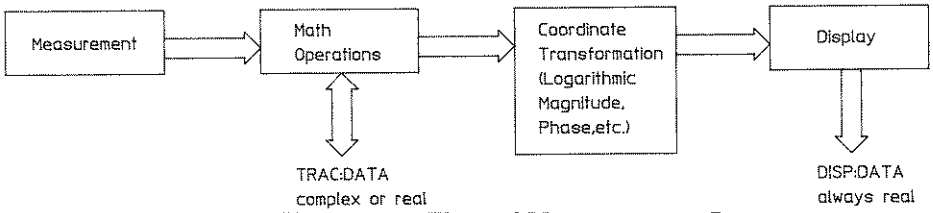


Figure 7-3. Flow of Measurement Data

After measurement data is collected, any specified math operations are performed. Data is then transformed into the specified coordinate system and sent to the display. TRAC:DATA provides access to the raw measurement data after math operations have been performed. This data can be either complex or real. DISP:DATA provides access to the displayed data, after the coordinate transformation. This data is always real.

NOTE Both TRAC:DATA and DISP:DATA allow you to take measurement data out of the analyzer. However, only TRAC:DATA allows you to put measurement data back into the analyzer.

With a few exceptions, trace commands must be directed to a trace in one of the two displays: A or B. The trace in display A is called trace A; the trace in display B is called trace B. To specify a trace, insert one of the following items between TRACE or TRAC and the rest of the command:

- :A – as in TRAC:A:RES FRES
- :B – as in TRACE:B:DTABLE:DATA?
- 1 – as in TRACE1:TITLE 'MyTrace'
- 2 – as in TRAC2:HEAD:XINC?

Using :A or 1 directs the command to trace A. Using :B or 2 directs the command to trace B. If you don't explicitly specify one of the traces, the command is directed to trace A.

TRAC:DATA[?] command/query

Overlapped: no
Delayed result: no
Pass control required: no
Power-up state: variable

Example Statements:
```
OUTPUT 711;"Trac:B:Data?"
```

Command Syntax: TRACe[<spec>]:DATA<sp><block_data>

<spec>::=~:A~|:B|1|2

<block_data> takes one of four forms. The form it takes for a particular transfer depends on:

- Whether the data is real or complex, and
- Whether the data is ASCII-encoded or binary-encoded

When data is real (TRAC:HEAD:YPO 1) and is ASCII-encoded (TRAC:HEAD:AFOR ASC):

<block_data>::={<point>,}...<point n><LF><^END>

<point>::=the y-axis values for the 1st through nth x-axis points (specify n with the TRAC:HEAD:POIN command)
Send y-axis values using the NRf format.

When data is complex (TRAC:HEAD:YPO 2) and is ASCII-encoded:

<block_data>::={<point>,}...<point n><LF><^END>

<point>::=<real_y>,<imag_y>

<real_y>::=the real part of y-axis values for the 1st through nth x-axis points

<imag_y>::=the imaginary part of y-axis values for the 1st through nth x-axis points

Send both real and imaginary y-axis values using the NRf format.

When data is real and is binary-encoded (TRAC:HEAD:AFOR FP32 or TRAC:HEAD:AFOR FP64):

<block_data>::=#<byte><length_bytes>{<point>}...

<byte>::=one ASCII-encoded byte that specifies the number of length bytes to follow

<length_bytes>::=ASCII-encoded bytes that specify the number of data bytes to follow

<point>::=the y-axis values for the 1st through nth x-axis points (specify n with the TRAC:HEAD:POIN command)

If TRAC:HEAD:AFOR FP32 is specified, y-axis values must be encoded as 32-bit binary floating point numbers. If TRAC:HEAD:AFOR FP64 is specified, y-axis values must be encoded as 64-bit binary floating point numbers.

When data is complex and is binary-encoded:

<block_data>::=#<byte><length_bytes>{<point>}...

<byte> and <length_bytes> have the same meaning as when data is real and binary-encoded.

<point>::=<real_y>,<imag_y>

<real_y>::=the real part of y-axis values for the 1st through nth x-axis points

<imag_y>::=the imaginary part of y-axis values for the 1st through nth x-axis points

If TRAC:HEAD:AFOR FP32 is specified, y-axis values must be encoded as 32-bit binary floating point numbers. If TRAC:HEAD:AFOR FP64 is specified, y-axis values must be encoded as 64-bit binary floating point numbers.

Query Syntax: TRACe[<spec>]:DATA?

Returned Format: <block_data>

Description:

This command lets you load a block of data into the specified trace.

Data is sent to the analyzer as a series of points. You can send as few as 3 and as many as 512 points if the data is in the frequency domain. You can send as few as 3 and as many as 1024 points if the data is in the time domain. When you send fewer than 512 points of frequency data, the points you send are used as the first points in the block and the remaining points are given a value of 0. The same thing is true when you send fewer than 512 points of complex time data or fewer than 1024 points of real time data.

The analyzer always displays 401 points of frequency-domain data. When you send a block of frequency data, the last 111 points are not displayed. When you send a block of zoomed frequency data (TRAC:HEAD:XOR ≠ 0), the last 56 points are actually lower in frequency than XOR.

NOTE Send 512 complex points to the analyzer if you plan to manipulate the data with the analyzer's waveform math capabilities.

For a block of baseband frequency data, the imaginary part of the 0 Hz bin actually contains the real part of the 513th bin. The analyzer uses this value when it performs an inverse FFT on the block.

There are two things you must do before you send a block of data to the analyzer with this command:

- Set up the trace that will receive the data so that it is domain-compatible (frequency or time) with the data you plan to send. This is done with the TRAC:RES command
- Use commands in the TRAC:HEAD subsystem to characterize the data you plan to send

The following commands are used to characterize a block of data points:

- TRAC:HEAD:AFOR – indicates whether the data is ASCII-encoded or binary-encoded
- TRAC:HEAD:POIN – indicates how many data points will be sent
- TRAC:HEAD:NAME – specifies a name for the data
- TRAC:HEAD:XINC – specifies the x-axis increment between data points (must be in Hz for frequency-domain traces, seconds for time-domain traces)
- TRAC:HEAD:XOR – specifies the x-axis value of the first data point (must be in Hz for frequency-domain traces, seconds for time-domain traces)
- TRAC:HEAD:XPO – ignored because the value must always be 0
- TRAC:HEAD:YINC – ignored when TRAC:HEAD:YPO is not 0
- TRAC:HEAD:YOR – ignored when TRAC:HEAD:YPO is not 0
- TRAC:HEAD:YPO – specifies the number of y-axis values to be sent with each data point (must be 1 or 2: 1 when data is real, 2 when data is complex)

Except for TRAC:HEAD:AFOR, all of the preceding commands have two special properties:

- The analyzer retains the values sent with these commands, but does not use them until you send a block of data points with the TRAC:DATA command
- You will not be able to query the value last sent with these commands. This is because the query forms of the commands always return the current values of the specified trace

For example, assume that you have sent TRAC:A:HEAD:NAME 'My_trace', but have not yet sent TRAC:A:DATA. Also assume that the currently displayed trace A data is named Spectrum Chan 1. In this case, the response to TRAC:A:HEAD:NAME? will be "Spectrum Chan 1". However, when you do send TRAC:A:DATA followed by a block of data points, the data will be displayed in trace A and will be given the name My_trace. In this case, the response to TRAC:A:HEAD:NAME? will be "My_trace".

The TRAC:DATA query returns the specified trace as a block of data points.

TRAC:DATA:SET[?] command/query

Overlapped: no
Delayed result: no
Pass control required: no
Power-up state: variable

Example Statements:

```
OUTPUT 711"TRAC:B:SET?"
```

Command Syntax: TRACe[<spec>]:DATA:SET<sp><block_data>

<spec>::=~:A~|:B|1|2

<block_data> takes one of two forms depending on whether you are sending ASCII-encoded or binary-encoded data. When you are sending ASCII-encoded data (TRAC:HEAD:AFOR ASC):

<block_data>::=#0{<line><LF>}...<^END>

<line>::=data on a particular line of the record

{<line><LF>} is repeated until all lines have been sent.

When you are sending binary-encoded data (TRAC:HEAD:AFOR FP32 or TRAC:HEAD:AFOR FP64):

<block_data>::=#<byte><length_bytes>{<data_byte>}...

<byte>::=one ASCII-encoded byte that specifies the number of length bytes to follow

<length_bytes>::=ASCII-encoded bytes that specify the number of data bytes to follow

<data_byte>::=one byte of the record's data

<data_byte> is repeated until the number of bytes specified in <length_bytes> has been sent.

Query Syntax: TRACe[<spec>]:DATA:SET?

Returned Format: <block_data>

Description:

Use this command to transfer a trace file from the controller to the analyzer via the HP-IB. A trace file is described in Chapter 4, "Transferring Data."

The query transfers a complete measurement result from the analyzer to the controller.

TRAC:HEADer — selector

Description:

This command only selects the TRAC:HEAD subsystem. Commands in this subsystem are used to define characteristics of the data you will send with the TRAC:DATA command. Queries in this subsystem are used to determine characteristics of the data returned by the TRAC:DATA query. Sending TRAC:HEAD alone does nothing.

TRAC:HEAD:AFORmat[?] — command/query

Overlapped: no
Delayed result: no
Pass control required: no
Power-up state: ASC

Example Statements:

```
OUTPUT 711;"Trac:B:Head:Afor ASC"
OUTPUT 711;"trace2:header:aformat FP64"
OUTPUT 711;"TRAC:HEAD:AFOR?"
```

Command Syntax: TRACe[<spec>]:HEADer:AFORmat<sp>{ASCii|FP32|FP64}

<spec>::=~:A~|:B|1|2

Query Syntax: TRACe[<spec>]:HEADer:AFORmat?

Returned Format: {ASC|FP32|FP64}<LF><^END>

Description:

Trace data can either be ASCII-encoded or binary-encoded when it is transferred between the analyzer and an external controller. Such transfers occur when you use TRAC:DATA or TRAC:DATA:SET. This command lets you specify how the trace data should be encoded.

NOTE Data encoding must be the same for both traces at any given time. So regardless of the trace you specify when you send this command, encoding for both is changed.

When ASC is selected, data is sent as a series of y-axis values separated by commas. The values are ASCII-encoded and are formatted as NRf decimal numbers.

FP32 and FP64 both specify binary encoding. When FP32 is selected, data is sent as a series of y-axis values within a definite length block. The values are encoded as 32-bit binary floating point numbers. When FP64 is selected, data is also sent as a series of y-axis values within a definite length block. However, the values are encoded as 64-bit binary floating point numbers.

The query returns ASC, FP32, or FP64, depending on the option currently specified.

TRAC:HEAD:NAME[?] command/query

Overlapped: no
Delayed result: no
Pass control required: no
Power-up state: "Spectrum Chan 1" (display A)
"Time Chan 1" (display B)

Example Statements:

```
OUTPUT 711;"Trac:Head:Name "TRACE DATA""
OUTPUT 711;"Trace:B:Header:Name "TRACE1""
OUTPUT 711;"trac:a:head:name?"
```

Command Syntax: TRACe[<spec>]:HEADer:NAME<sp><trace_name>

<spec>::=~:A~ |:B|1|2

<trace_name>::=0 to 30 printable ASCII characters.

Query Syntax: TRACe[<spec>]:HEADer:NAME?

Returned Format: "<trace_name>"<LF><^END>

Description:

This command specifies a name for the data you will send with the TRAC:DATA command. The name will appear near the lower-left corner of the specified trace after you send a block of data points. To change the name of the trace currently displayed, you must use the TRAC:TITL command.

The query returns the current name of the specified trace. It does not return the last value you sent with the TRAC:HEAD:NAME command. See TRAC:DATA for more information.

TRAC:HEAD:POINts[?] command/query

Overlapped: no
Delayed result: no
Pass control required: no
Power-up state: 512 (display A)
1024 (display B)

Example Statements:

```
OUTPUT 711;"TRAC2:HEAD:POIN 512"
OUTPUT 711;"Trace1:Header:Points 1024"
OUTPUT 711;"Trac:a:Head:Poin?"
```

Command Syntax: TRACe[<spec>]:HEADer:POINts<sp><value>

<spec>::=~:A~|:B|1|2

<value>::=an integer from 3 through 512 for all frequency-domain displays

an integer from 3 through 512 for time-domain displays containing complex data

an integer from 3 through 1024 for time-domain displays containing real data

All values should be sent using the NRf format.

Query Syntax: TRACe[<spec>]:HEADer:POINts?

Returned Format: <value><LF><^END>

<value>::=an integer (NR1 format)

Description:

Use this command to indicate how many data points you will send to the specified trace with the TRAC:DATA command.

You can send a different number of data points than you have specified with this command. The analyzer will accept the points if the total number of points is within the specified range for <value>. Using the TRAC:HEAD:POIN command simply allows you to confirm that the total number of data points you will send is within the specified range.

The response to this query tells you how many data points will be returned from the specified trace in response to the TRAC:DATA query. It does not tell you the value last sent with the TRAC:HEAD:POIN command. See TRAC:DATA for more information.

TRAC:HEAD:PREamble? query

Overlapped: no
Delayed result: no
Pass control required: no
Power-up state: variable

Example Statement:

```
OUTPUT 711;"trac:b:head:pre?"
OUTPUT 711;"TRACE2:HEADER:PREAMBLE?"
```

Query Syntax: TRACe[<spec>]:HEADer:PREamble?

<spec>::=~:A~|:B|1|2

Returned Format: <points>,<x_per_point>,<x_origin>,<x_increment>,<y_per_point>,<y_origin>,<y_increment><LF><^END>

<points>::=number of discrete points on the trace's x-axis (same as returned with TRAC:HEAD:POIN?)

<x_per_point>::=number of x-axis values per point (same as returned with TRAC:HEAD:XPO?)

<x_origin>::=x-axis value of the first displayed point (same as returned with TRAC:HEAD:XOR?)

<x_increment>::=increment between x-axis points (same as returned with TRAC:HEAD:XINC?)

<y_per_point>::=number of y-axis values per point (same as returned with TRAC:HEAD:YPO?)

<y_origin>::=y-axis value of the lowest point on the specified trace (same as returned with TRAC:HEAD:YOR?)

<y_increment>::=optimum y-axis value per division (same as returned with TRAC:HEAD:YINC?)

<points>, <x_per_point>, and <y_per_point> are integers (NR1 format). All other values are decimal numbers (NR2 or NR3 format).

Description:

This query returns seven pieces of information separated by commas. The information is useful for setting up an array to receive trace data (returned from TRAC:DATA?).

NOTE As the Returned Format indicates, each piece of information can be returned separately in response to its own query.

The <points>, <x_per_point>, and <y_per_point> values are used together to tell you how many values you must read after sending the TRAC:DATA query. The formula is:

# of values to read = <points>×(<x_per_point+<y_per_point>)

The analyzer does not return x-axis values for each data point. Instead, it provides <x_origin> and <x_increment> values so you can assign an x-axis value to each returned point. <x_origin> is the x-axis value for the first point. Add <x_increment> to the first point's x-axis value to get the value of the second point. Add <x_increment> to the second point's x-axis value to get the value of the third point and so on.

The values returned in <y_origin> and <y_increment> should be ignored when the value of <y_points> is something other than 0 (zero).

TRAC:HEAD:XINCrement[?] command/query

Overlapped: no
Delayed result: no
Pass control required: no
Power-up state: 256 (display A)
3.81E-6(display B)

Example Statements:

```
OUTPUT 711;"TRAC2:HEAD:XINC 100"
OUTPUT 711;"TRACE:A:HEADER:XINCREMENT 125"
OUTPUT 711;"Trac:B:Head:Xinc?"
```

Command Syntax: TRACe[<spec>]:HEADer:XINCrement<sp><value>

<spec>::=~:A~|:B|1|2

<value>::=a decimal number (NRf format)

Must be in Hz for frequency-domain traces, seconds for time domain traces.

Query Syntax: TRACe[<spec>]:HEADer:XINCrement?

Returned Format: <value><LF><^END>

Description:

Use this command to specify the x-axis increment between the data points you will send with the TRAC:DATA command.

The analyzer uses TRAC:HEAD:XINC and TRAC:HEAD:XOR together to assign x-axis values to the data points you send with the TRAC:DATA command. It uses TRAC:HEAD:XOR for the x-axis value of the first point. It adds TRAC:HEAD:XINC to the first point's x-axis value to get the value of the second point. It adds TRAC:HEAD:XINC to the second point's x-axis value to get the value of the third point and so on.

Use TRAC:HEAD:XUN? to determine the unit that will be assumed for the value you send.

The response to this query tells you the current x-axis increment between points on the specified trace. It does not tell you the value last sent with the TRAC:HEAD:XINC command. See TRAC:DATA for more information.

TRAC:HEAD:XNAMe? query

Overlapped: no
Delayed result: no
Pass control required: no
Power-up state: "Frequency" (display A)
"Time" (display B)

Example Statement:

```
OUTPUT 711;"Trac:B:Head:Xnam?"
OUTPUT 711;"trace1:header:xname?"
```

Query Syntax: TRACe[<spec>]:HEADer:XNAMe?

<spec>::=~:A~|:B|1|2

Returned Format: "{Frequency|Time}"<LF><^END>

Description:

This query returns the name of the specified trace's x-axis. The name tells you whether the displayed data is in the frequency or the time domain.

TRAC:HEAD:XORigin[?] command/query

Overlapped: no
Delayed result: no
Pass control required: no
Power-up state: 0

Example Statements:

```
OUTPUT 711;"Trac2:Head:Xor 10"
OUTPUT 711;"Trace:B:Header:Xorigin 250"
OUTPUT 711;"trac1:head:xor?"
```

Command Syntax: TRACe[<spec>]:HEADer:XORigin<sp><value>

<spec>::=~:A~|:B|1|2

<value>::=a decimal number (NRf format)

Query Syntax: TRACe[<spec>]:HEADer:XORigin?

Returned Format: <value><LF><^END>

Description:

Use this command to specify the x-axis value of the first displayed data point you will send with the TRAC:DATA command.

The analyzer uses TRAC:HEAD:XOR and TRAC:HEAD:XINC together to assign x-axis values to the data points you send with the TRAC:DATA command. It uses TRAC:HEAD:XOR for the x-axis value of the first point. It adds TRAC:HEAD:XINC to the first point's x-axis value to get the value of the second point. It adds TRAC:HEAD:XINC to the second point's x-axis value to get the value of the third point and so on.

Use TRAC:HEAD:XUN? to determine the unit that will be assumed for the value you send.

The response to this query tells you the current x-axis value of the first point on the specified trace. It does not tell you the value last sent with the TRAC:HEAD:XOR command. See TRAC:DATA for more information.

TRAC:HEAD:XPOints[?] command/query

Overlapped: no
Delayed result: no
Pass control required: no
Power-up state: 0

Example Statement: `OUTPUT 711;"Trac2:Head:Xpo?"`

Command Syntax: TRACe[<spec>]:HEADer:XPOints<sp><value>

<spec>::=~:A~|:B|1|2

<value>::=an integer (NRf format)

Query Syntax: TRACe[<spec>]:HEADer:XPOints?

Returned Format: 0<LF><^END>

Description:

Any value you send with this command is ignored. The analyzer always assumes that the value of TRAC:HEAD:XPO is 0. This means that data points sent with the TRAC:DATA command can not contain any x-axis values. Instead, x-axis values will be calculated using the values you sent with the TRAC:HEAD:XINC and TRAC:HEAD:XOR commands.

The TRAC:HEAD:XPO query always returns 0.

TRAC:HEAD:XUNits? query

Overlapped: no
Delayed result: no
Pass control required: no
Power-up state: "HZ" (display A)
"S" (display B)

Example Statement: `OUTPUT 711;"trac:b:head:xun?"`

Query Syntax: TRACe[<spec>]:HEADer:XUNits?

<spec>::=~:A~|:B|1|2

Returned Format: "{HZ|S}"<LF><^END>

Description:

This query tells you what units apply to the TRAC:HEAD:XINC and TRAC:HEAD:XOR values.

TRAC:HEAD:YINCrement[?] command/query

Overlapped: no
Delayed result: no
Pass control required: no
Power-up state: variable

Example Statement: `OUTPUT 711;"Trac1:Head:Yinc?"`

Command Syntax: TRACe[<spec>]:HEADer:YINCrement<sp><value>

<spec>::=~:A~|:B|1|2

<value>::=a decimal number (NRf format)

Query Syntax: TRACe[<spec>]:HEADer:YINCrement?

Returned Format: <value><LF><^END>

Description:

Any value you send with this command is ignored. The analyzer can only use the value you send with this command if the value of TRAC:HEAD:YPO is 0. Since the only allowable values of TRAC:HEAD:YPO are 1 and 2, TRAC:HEAD:YINC can never be used.

This query returns the optimum y-axis value per division for the specified trace. The value returned is the result of the following calculation:

$$(Ymax - Ymin)/8$$

Where:

Ymax = the y-axis value of the highest point on the trace

Ymin = the y-axis value of the lowest point on the trace

The value is returned in the current y-axis units.

TRAC:HEAD:YNAMe? query

Overlapped: no
Delayed result: no
Pass control required: no
Power-up state: ""

Example Statement:
```
OUTPUT 711;"Trac:Head:Ynam?"
OUTPUT 711;"trace1:header:yname?"
```

Query Syntax: TRACe[<spec>]:HEADer:YNAMe?

<spec>::=~:A~|:B|1|2

Returned Format: {""}<LF><^END>

Description:

This query returns the null string.

TRAC:HEAD:YORigin[?] command/query

Overlapped: no
Delayed result: no
Pass control required: no
Power-up state: variable

Example Statement: `OUTPUT 711;"Trac:A:Head:Yor?"`

Command Syntax: TRACe[<spec>]:HEADer:YORigin<sp><value>

<spec>::=~:A~|:B|1|2

<value>::=a decimal number (NRf format)

Query Syntax: TRACe[<spec>]:HEADer:YORigin?

Returned Format: <value><LF><^END>

<value>::=a decimal number (NRf format)

Description:

Any value you send with this command is ignored. The analyzer can only use the value you send with this command if the value of TRAC:HEAD:YPO is 0. Since the only allowable values of TRAC:HEAD:YPO are 1 and 2, TRAC:HEAD:YOR can never be used.

The query returns the y-axis value of the lowest point on specified trace.

TRAC:HEAD:YPOints[?] command/query

Overlapped: no
Delayed result: no
Pass control required: no
Power-up state: 2 (display A)
1 (display B)

Example Statements:

```
OUTPUT 711;"TRAC1:HEAD:YPO 1"
OUTPUT 711;"Trace2:Header:Ypoints 2"
OUTPUT 711;"Trac:B:Head:Ypo?"
```

Command Syntax: TRACe[<spec>]:HEADer:YPOints<sp><value>

<spec>::=~:A~|:B|1|2

<value>::=1 if the data is real

2 if the data is complex

Query Syntax: TRACe[<spec>]:HEADer:YPOints?

Returned Format: {1|2}<LF><^END>

Description:

The TRAC:DATA command lets you send a block of data points to the analyzer. Each point will consist of either one or two y-axis values, depending on whether the data is real or complex. Before sending a block of points, you must use this command (TRAC:HEAD:YPO) to tell the analyzer how many y-axis values will be sent with each point.

The response to this query tells you how many y-axis values will be returned from the specified trace in response to the TRAC:DATA query. It does not tell you the value last sent with the TRAC:HEAD:YPO command. See TRAC:DATA for more information.

TRAC:HEAD:YUNits? query

Overlapped: no
Delayed result: no
Pass control required: no
Power-up state: "V"

Example Statement: `OUTPUT 711;"trac:a:head:yun?"`

Query Syntax: TRACe[<spec>]:HEADer:YUNits?

<spec>::=~:A~|:B|1|2

Returned Format: "[<unit>]"<LF><^END>

<unit>::=V|V2|V2/HZ

Description:

This query tells you what unit applies to the y-axis values returned from the TRAC:DATA query.

NOTE Not listed in Returned Format are the many special units that can result from math operations or the application of engineering units. However, such units are also valid responses.

TRAC:DATA returns the raw measurement data from which display data is derived. The unit used for the raw data is dependent on the selected measurement results (TRAC:RES), but not the selected display units (DISP:Y:SCAL:UNIT). As a result, the unit for TRAC:DATA values may not be the same as the unit shown on the current display.

TRAC:RESult[?] command/query

Overlapped: no
Delayed result: yes
Pass control required: no
Power-up state: SPEC1 (display A)
TIME1 (display B)

Example Statements:

```
OUTPUT 711;"TRAC1:RES COH"
OUTPUT 711;"TRACE:B:RESULT F5"
OUTPUT 711;"Trac2:Res?"
```

Command Syntax: TRACe[<spec>]:RESult<sp><meas_data>

<spec>::=~:A~|:B|1|2

<meas_data>::=COHerence|CSPectrum|F<num>|FRESponse|K<num>
|PSD{1|2}|SPECtrum{1|2}|TIME{1|2}

<num>::=1|2|3|4|5

Query Syntax: TRACe[<spec>]:RESult?

Returned Format: <meas_data><LF><^END>

<meas_data>::=COH|CSP|F<num>|FRES|K<num>|PSD{1|2}|SPEC{1|2}|TIME{1|2}

Description:

Use this command to select one of the measurement results, math functions or math constants. Your selection is displayed in the specified trace. Special restrictions on the options you can select follow.

- Coherence (TRAC:RES COH) is only available when the analyzer is in the two-channel mode (CONF:TYPE NETW) and the measurement is averaged (AVER:STAT ON). It is never available when the average type is peak hold (AVER:TYPE PEAK).
- The cross spectrum (TRAC:RES CSP) is only available when the analyzer is in the two-channel mode. It is never available when the average type is peak hold.
- The math functions (TRAC:RES F<num>) are only available if they have already been defined and if all of the data they require is currently available to the analyzer. You can define math functions using the USER:EXPR command.
- Frequency response (TRAC:RES FRES) is only available when the analyzer is in the two-channel mode. It is never available when the average type is peak hold.
- The math constants (TRAC:RES K<num>) are available at all times. They are defined using the USER:VAR commands.
- Power spectral density for channel 1 (TRAC:RES PSD1) is available at all times. Power spectral density for channel 2 (TRAC:RES PSD2) is only available when the analyzer is in the two-channel mode.

- The spectrum for channel 1 (TRAC:RES SPEC1) is available at all times. The spectrum for channel 2 (TRAC:RES SPEC2) is only available when the analyzer is in the two-channel mode. The spectra are linear spectra if averaging is off or if vector averaging is on. They are power spectra if rms averaging is on.
- The time record for channel 1 (TRAC:RES TIME1) is available at all times. The time record for channel 2 (TRAC:RES TIME2) is only available when the analyzer is in the two-channel mode. Time records are uncalibrated.

The query response tells you which result, function or constant is currently displayed.

TRAC:TITLe[?] command/query

Overlapped: no
Delayed result: no
Pass control required: no
Power-up state: **

Example Statements:

```
OUTPUT 711;"Tracl:Titl ""TEST1"""
OUTPUT 711;"trace:b:title ""TRACE_MAG"""
OUTPUT 711;"TRAC2:TITL?"
```

Command Syntax: TRACe[<spec>]:TITLe<sp>"<name>"

<spec>::=~:A~|:B|1|2

<name>::=0 to 30 printable ASCII characters.

Query Syntax: TRACe[<spec>]:TITLe?

Returned Format: "<name>"<LF><^END>

Description:

This command allows you to enter a user-specified name for the displayed data. The name you enter will appear near the lower-left of the specified trace.

The analyzer assigns a default name to the various measurement results you can display. These default names are related to the option selected with the TRAC:RES command. When you send a name with the TRAC:TITL command, that name replaces the default name. If you later decide that you prefer the default name to the name you have assigned, just send this command with the null string (TRAC:TITL ""). The default name will reappear in the lower-left corner of the trace.

The query returns the current user-specified name for the specified trace.

TRIGger subsystem

Description:

This subsystem contains commands related to the analyzer's triggering functions. See the ARM subsystem for commands related to trigger arming.

TRIG:DELay[?] command/query

Overlapped: no
Delayed result: yes
Pass control required: no
Power-up state: 0

Example Statements:

```
OUTPUT 711;"Trig:Dell -7MS"
OUTPUT 711;"Trigger:Delay2 1S"
OUTPUT 711;"trig:del2?"
```

Command Syntax: TRIGger:DELay[~1~|2]<sp><value>[<unit>]

<value>::=a decimal number (NRf format)

<unit>::=~S~|MS|US

Query Syntax: TRIGger:DELay[~1~|2]?

Returned Format: <value><LF><^END>

Description:

This command allows you to enter a time delay between two points: the point at which the analyzer is triggered and the point at which the specified channel starts collecting data. This delay is also referred to as trigger delay.

If you want the channel to start collecting data before the trigger point, you must specify a pre-trigger delay. This is done by sending a negative value with the command. If you want the channel to start collecting data after the trigger point, you must specify a post-trigger delay. This is done by sending a positive value with the command.

The maximum pre-trigger delay is 8 time records. The maximum post-trigger delay is 8191 seconds. Also, there can be no more than 7 time records difference between the delay specified for channel 1 and the delay specified for channel 2. You can determine the current time record length with the SWE:TIME query.

You can either use numbers or one of two nonnumeric parameters to set the value of TRIG:DEL. The nonnumeric parameters are:

- UP – increases the current value of TRIG:DEL
- DOWN – decreases the current value of TRIG:DEL

The amount of increase or decrease is equal to the increment between x-axis points for the current time record.

The query returns the delay currently selected for the specified channel. The value is returned in seconds.

TRIG[:IMMediate] command

Overlapped: no
Delayed result: no
Pass control required: no
Power-up state: not applicable

Example Statements:

```
OUTPUT 711;"TRIG"
OUTPUT 711;"Trigger:Immediate"
```

Command Syntax: TRIGger[:IMMediate]

Description:

This command triggers the analyzer if the following two things are true:

- The trigger source must be the HP-IB (TRIG:SOUR BUS)
- The analyzer must be ready to trigger. (Bit 2 of the Device Status condition register must be set.)
- TRIG:IMM has the same effect as the *TRG command

TRIG:LEVel[?] command/query

Overlapped: no
Delayed result: yes
Pass control required: no
Power-up state: 0

Example Statements:

```
OUTPUT 711;"trig:lev .25"
OUTPUT 711;"TRIGGER:LEVEL 25PCT"
OUTPUT 711;"TRIG:LEV?"
```

Command Syntax: TRIGger:LEVel<sp>{{<percent>PCT}|<fraction>

<percent>::=an integer from -100 to 100

<fraction>::=a decimal number from -1 to 1 in increments of 0.01

Both <percent> and <fraction> are sent in NRf format.

Query Syntax: TRIGger:LEVel?

Returned Format: <fraction><LF><^END>

<fraction>::=a decimal number in NR1 or NR2 format

Description:

Use this command to specify the level at which an input signal can cause the analyzer to trigger. The value you send with this command is only used if the trigger source is one of the analyzer's two input channels (TRIG:SOUR INT1 or TRIG:SOUR INT2).

You can specify the trigger level either as a percentage or as a fraction of the trigger channel's current input range. (Use the INP:RANG command to determine the current range.) Sending a positive value causes the analyzer to trigger on the positive portion of the signal. Sending a negative value causes the analyzer to trigger on the negative portion of the signal.

You can either send a number or one of two nonnumeric parameters to set the trigger level. If you send a number, it is rounded to the nearest allowable percentage (an integer between -100 and 100). The nonnumeric parameters are:

- UP – increases the current trigger level by 1%
- DOWN – decreases the current trigger level by 1%

The query returns the trigger level currently specified. The value is returned in the fractional form.

TRIG:SLOPe[?] command/query

Overlapped: no
Delayed result: yes
Pass control required: no
Power-up state: POS

Example Statements:

```
OUTPUT 711;"TRIG:SLOP POS"
OUTPUT 711;"TRIGGER:SLOPE NEGATIVE"
OUTPUT 711;"Trig:Slop?"
```

Command Syntax: TRIGger:SLOPe<sp>{POSitive|NEGative}

Query Syntax: TRIGger:SLOPe?

Returned Format: {POS|NEG}<LF><^END>

Description:

The analyzer can trigger either when a trigger signal's slope is positive or when it is negative. Use this command to specify the slope on which the analyzer can trigger.

The trigger slope setting is only used for the following three trigger signals:

- The channel 1 input signal (TRIG:SOUR INT1)
- The channel 2 input signal (TRIG:SOUR INT2)
- The external trigger signal (TRIG:SOUR EXT)

The query returns the slope currently specified.

TRIG:SOURce[?] command/query

Overlapped: no
Delayed result: yes
Pass control required: no
Power-up state: FREE

Example Statements:

```
OUTPUT 711;"Trig:Sour Sour"
OUTPUT 711;"trigger:source internal1"
OUTPUT 711;"TRIG:SOUR?"
```

Command Syntax: TRIGger:SOURce<sp><option>

<option>::={BUS|FREErun|EXTernal|INTernal{1|2}|SOURce}

Query Syntax: TRIGger:SOURce?

Returned Format: {BUS|FREE|EXT|INT{1|2}|SOUR}<LF><^END>

Description:

The analyzer can be triggered from a number of different sources. This command allows you to specify the source of the trigger signal.

NOTE The analyzer can only be triggered if it is ready to trigger. Bit 2 of the Device Status condition register is set to 1 when the analyzer is ready to trigger.

A description of the options available for this command follows:

- BUS – The HP-IB acts as the trigger source. When this option is selected, the analyzer can be triggered with either the TRIG:IMM command or the *TRG command.
- EXTernal – The analyzer's external trigger BNC (on the rear panel) acts as the trigger source. With this option selected, the analyzer can be triggered when the signal at that BNC makes either a low-to-high or a high-to-low TTL transition. The setting of TRIG:SLOP determines which transition will trigger the analyzer.
- FREErun – The analyzer requires no trigger signal. It triggers itself as soon as it is armed.
- INTernal{1|2} – The specified input channel acts as the trigger source. With this option selected, the analyzer can be triggered when the input signal matches the settings of TRIG:LEV and TRIG:SLOP.
- SOURce – The analyzer's signal source acts as the trigger source. With this option selected, the analyzer can be triggered when a synchronization signal is generated by the source board.

The query returns the currently specified trigger source.

USER subsystem

Description:

Commands in this subsystem allow you to define the analyzer's math functions and constants. If HP Instrument BASIC is installed in the analyzer, additional commands are added to this subsystem. For information on these commands, see Appendix D in the HP Instrument BASIC Programming Reference.

USER:EXPRession[?] command/query

Overlapped: no
Delayed result: yes
Pass control required: no
Power-up state: ()

Example Statements:

```
OUTPUT 711;"User:Expr F1,(SPEC1*FFT(TIME2))"
OUTPUT 711;"User:Expr F5,(('MY_FILE'+K2)*JOM)"
OUTPUT 711;"user:expr? F4"
```

Command Syntax: USER:EXPRession<sp>F<num>,(<expr>)

<num>::=1|2|3|4|5

<expr>::=one or more of the following elements in a mathematical equation:

SPEC1, SPEC2, PSD1, PSD2, TIME1, TIME2, FRES, COH, CSP, F1:F5, K1:K5, '<filename>',JOM(, CONJ(, MAG(, REAL(, IMAG(, SQRT(, FFT(, IFFT(, and ()+*/

<filename>::=the name of a file containing data you want to include in the expression

Query Syntax USER:EXPRession?<sp>F<num>

Returned Format: (<expr>)<LF><^END>

Description:

Use this command to define any of the analyzer's five math functions.

The elements CONJ, MAG, REAL, IMAG, SQRT, FFT, and IFFT are all function operators. Use the following syntax to specify what data they are to operate on:

operator(data)

For example, if you want function 1 to take the inverse-FFT of the channel 1 frequency response, send the command USER:EXPR F1,(IFFT(FRES)). See the HP 35660A Getting Started Guide for more information on defining functions.

The query returns the current definition of the specified function.

USER:VARiable selector

Description:

This command only selects the USER:VAR subsystem. Sending USER:VAR alone does nothing.

The four commands in this subsystem can be thought of as two pairs of commands. Each pair allows you to define the specified constant. USER:VAR:IMAG and USER:VAR:REAL form one pair, while USER:VAR:MAGN and USER:VAR:PHAS form the other.

When you change the value of one member of a pair, the value of the other member of that pair remains constant. However, the value of both members of the other pair are changed so that all of the following formulas are true:

- imag=sin(phase)×magn
- real=cos(phase)×magn
- magn=$\sqrt{(imag^2+real^2)}$
- phase=arctan(imag/real)

USER:VAR:IMAGinary[?] command/query

Overlapped: no
Delayed result: yes
Pass control required: no
Power-up state: 0

Example Statements:

```
OUTPUT 711;"USER:VAR:IMAG K1,2"
OUTPUT 711;"User:Variable:Imaginary K3,1.41421"
OUTPUT 711;"User:Var:Imag? K4"
```

Command Syntax: USER:VARiable:IMAGinary<sp>K<num>,<value>

<num>::=1|2|3|4|5

<value>::=any x, where −340.28E+36≤x≤340.28E+36 (NRf format)

Query Syntax: USER:VARiable:IMAGinary?<sp>K<num>

Returned Format: <value><LF><^END>

Description:

Use this command to define the imaginary part of the specified math constant.

This command and USER:VAR:REAL are used together to completely define a constant. Once you have defined the real and imaginary parts of a constant, you can determine its magnitude and phase with the USER:VAR:MAGN and USER:VAR:PHAS queries. See USER:VAR for more information.

The query returns the current value of the specified constant's imaginary part.

USER:VAR:MAGNitude[?] command/query

Overlapped: no
Delayed result: yes
Pass control required: no
Power-up state: 1

Example Statements:

```
OUTPUT 711;"user:var:magn K5,5"
OUTPUT 711;"USER:VARIABLE:MAGNITUDE K2,7.43"
OUTPUT 711;"USER:VAR:MAGN? K3"
```

Command Syntax: USER:VARiable:MAGNitude<sp>K<num>,<value>

<num>::=1|2|3|4|5

<value>::=a decimal number from 0 to 340.28E36 (NRf format)

Query Syntax: USER:VARiable:MAGNitude?<sp>K<num>

Returned Format: <value><LF><^END>

Description:

Use this command to define the magnitude of the specified math constant.

This command and USER:VAR:PHAS are used together to completely define a constant. Once you have defined the magnitude and phase of a constant, you can determine the values of its real and imaginary parts with the USER:VAR:REAL and USER:VAR:IMAG commands. See USER:VAR for more information.

The query returns the current magnitude of the specified constant.

USER:VAR:PHASe[?] command/query

Overlapped: no
Delayed result: yes
Pass control required: no
Power-up state: 0

Example Statements:

```
OUTPUT 711;"USER:VAR:PHAS K1,45DEG"
OUTPUT 711;"USER:VARIABLE:PHASE K4,0.52RAD"
OUTPUT 711;"USER:VAR:PHAS? K5"
```

Command Syntax: USER:VARiable:PHASe<sp>K<num>,<value><unit>

<num>::=1|2|3|4|5

<value>::=a decimal number from −180 to 180 (when <unit> is DEG)
a decimal number from −3.1416 to 3.1416 (when <unit> is RAD)
The value is always sent in NRf format.

<unit>::={DEGrees|RADians}

Query Syntax: USER:VARiable:PHASe?<sp>K<num>

Returned Format: <value><LF><^END>

Description:

Use this command to define the phase of the specified math constant.

This command and USER:VAR:MAGN are used together to completely define a constant. Once you have defined the magnitude and phase of a constant, you can determine the values of its real and imaginary parts with the USER:VAR:REAL and USER:VAR:IMAG commands. See USER:VAR for more information.

The query returns the current phase of the specified constant. The value is always returned in the phase unit last used.

USER:VAR:REAL[?] command/query

Overlapped: no
Delayed result: yes
Pass control required: no
Power-up state: 1

Example Statements:

```
OUTPUT 711;"User:Var:Real K3,6.7"
OUTPUT 711;"user:variable:real K1,0.70711"
OUTPUT 711;"USER:VAR:REAL? K5"
```

Command Syntax: USER:VARiable:REAL<sp>K<num>,<sp><value>

<num>::={1|2|3|4|5}

<value>::=any x, where $-340.28E+36 \leq x \leq 340.28E+36$ (NRf format)

Query Syntax: USER:VARiable:REAL?<sp>K<num>

<num> is defined in Command Syntax.

Returned Format: <value><LF><^END>

<value>::=decimal number in NRf format.

Description:

Use this command to define the real part of the specified math constant.

This command and USER:VAR:IMAG are used together to completely define a constant. Once you have defined the real and imaginary parts of a constant, you can determine its magnitude and phase with the USER:VAR:MAGN and USER:VAR:PHAS queries. See USER:VAR for more information.

The query returns the current value of the specified constant's real part.

WINDow — subsystem

Description:

Commands in this subsystem are used to define and select windowing functions.

WIND:CONStant — selector

Description:

This command just selects the WIND:CONS subsystem. Sending WIND:CONS alone does nothing.

WIND:CONS:EXPonential[?] command/query

Overlapped: no
Delayed result: no
Pass control required: no
Power-up state: 9999

Example Statements:

```
OUTPUT 711;"WIND2:CONS:EXP .007"
OUTPUT 711;"Window1:Constant:Exponential 3.5MS"
OUTPUT 711;"Wind:Cons:Exp?"
```

Command Syntax: WINDow[~1~|2]:CONStant:EXPonential<sp><value>[<unit>]

<value>::=a decimal number from 100E-9 through 9.999E6 (when the unit is seconds)

Send the value using the NRf format.

<unit>::=~S~|MS|US

Query Syntax: WINDow[~1~|2]:CONStant:EXPonential?

Returned Format: <value><LF><^END>

Description:

Use this command to specify a rate of decay for the specified channel's Exponential window.

The value of an Exponential window is 1.0 at the beginning of a time record. The value then decays at an exponential rate determined by the following formula:

$$e^{(-t/TC)}$$

Where:

t= time record length (SWE:TIME)

TC=time constant entered with WIND:CONS:EXP

The value you send with this command can also determine the rate of decay for the other channel's Force window. See WIND:TYPE for more information.

You can step the current value of this constant up or down by sending WIND:CONS:EXP UP or WIND:CONS:EXP DOWN.

The query returns the current value of the specified time constant. The value is returned in seconds.

WIND:CONS:FORCe[?] command/query

Overlapped: no
Delayed result: no
Pass control required: no
Power-up state: 9999

Example Statements:

```
OUTPUT 711;"WIND2:CONS:FORC .007"
OUTPUT 711;"Window1:Constant:Force 3.5MS"
OUTPUT 711;"Wind:Cons:Forc?"
```

Command Syntax: WINDow[~1~|2]:CONStant:FORCe<sp><value>[<unit>]

<value>::=a decimal number from 100E-9 through 9.999E6 (when the unit is seconds)

Values are sent using the NRf format.

<unit>::=~S~|MS|US

Query Syntax: WINDow[~1~|2]:CONStant:FORCe?

Returned Format: <value><LF><^END>

Description:

Use this command to indicate how long the specified channel's Force window should pass the input signal. See WIND:TYPE for more information.

You can step the current value of this constant up or down by sending WIND:CONS:FORC UP or WIND:CONS:FORC DOWN.

The query returns the current value specified for this constant. The value is returned in seconds.

WIND[:TYPE][?] command/query

Overlapped: no
Delayed result: yes
Pass control required: no
Power-up state: FLAT

Example Statements:
```
OUTPUT 711;"Wind Flat"
OUTPUT 711;"Window2:Type Exponential"
OUTPUT 711;"wind1:type?"
```

Command Syntax: WINDow[~1~|2][:TYPE]<sp>
{FLATtop|HANNing|UNIForm|FORCe|EXPonential}

Query Syntax: WINDow[~1~|2][:TYPE]?

Returned Format: {FLAT|HANN|UNIF|FORC|EXP}<LF><^END>

Description:

Use this command to select a windowing function (also called a window) for the input data.

A window is a time-domain weighting function that is applied to input signals. It reduces the negative effects of signals that are not periodic within the time record. For more information, see the HP 35660A Getting Started Guide.

When you select the Flat Top, Hanning, or Uniform window, the window is applied to both channels. Only the Force and Exponential windows can be applied to a single channel. A description of the windows follows:

- EXPonential – Selects the Exponential window. This window decays at an exponential rate. The rate is determined by the following formula:

$$e^{(-t/TC)}$$

Where:

t=time record length (SWE:TIME)

TC=time constant entered with WIND:CONS:EXP

This window is useful when you are measuring the response of a system that was excited by an impulse. If the system is lightly damped, you can ensure that the response decays within the time record by entering a time constant that is less than or equal to 1/4 the time record length.

- FLATtop – Selects the Flat Top window. This window has greater amplitude accuracy, but lower frequency resolution than the Hanning window. It is especially useful when you are measuring the amplitude of fixed-sine signals.
- FORCe – Selects the Force window. This window passes the input signal for a fixed amount of time (specified with the WIND:CONS:FORC command). After that time, the signal is set to its average value for the remainder of the time record. The Force window is most useful when you are measuring the excitation force of an impact test.

  After the Force window is applied to a channel's input signal, the signal may also be attenuated by an amount that decreases exponentially. This is the case only when the Exponential window is applied to the other channel's input signal. The rate of decrease is determined by the setting of WIND:CONS:EXP for that other channel.
- HANNing – Selects the Hanning window (sometimes called the Hann or Random window). This window offers greater frequency resolution, but lower amplitude accuracy than the Flat Top window. It is most useful when you are measuring random noise signals.
- UNIForm – Selects the Uniform window (sometimes called the Transient window). This window does not attenuate any portion of the time record, so it is like using no window at all. The Uniform window should only be selected when you are analyzing signals that are periodic within the time record. The analyzer's periodic chirp is one such signal.

The query indicates which window is currently being applied to the specified input channel.

Appendix A
Selecting Units

The following tables (Tables A-1 through A-4) show the units available for the y-axis. These units are specified with the HP-IB commands shown below. See "Command Reference" (in this manual) for a description of these commands.

- Y-axis start (bottom), stop (top) or center reference:
  DISP[~:A~ |:B|1|2][:Y]:SCAL:STAR < sp >< value >[< units >]
  DISP[~:A~ |:B|1|2][:Y]:SCAL:STOP < sp >< value >[< units >]
  DISP[~:A~ |:B|1|2][:Y]:SCAL:CENT < sp >< value >[< units >]
- Y-axis (vertical) units:
  DISP[~:A~ |:B|1|2][:Y]:SCAL:UNIT< sp >{'|"}< units >{'|"}
- Y-axis (vertical) per division:
  DISP[~:A~ |:B|1|2][:Y]:SCAL:DIV< sp >< value >[< units >]

Use Table A-1 to determine the units you may use if you are setting the y-axis start, stop or center reference or the y-axis units. Use Table A-2 to determine the units you may use if you are setting the y-axis per division. If you are using engineering units, use Tables A-3 and A-4 instead of Tables A-1 and A-2.

NOTE If you do not specify units, the HP 35660A analyzer uses the default unit shown as bold type in Tables A-1 through A-4.

The units that you may specify depend on the measurement you select and the trace type. If you do not know the current measurement selection you can send the query:

TRAC[~:A~ |:B|1|2]:RES?

or select a measurement with:

TRAC[~:A~ |:B|1|2]:RES< sp >< meas >

where:

< meas >::= {SPEC{1|2}|PSD{1|2}|TIME{1|2}|COHerence|FRESponse
|CSPectrum|F< num >|K< num >}.

< num >::= {1|2|3|4|5}

Likewise, you can query for the trace type with:

DISP[~:A~ |:B|1|2][:Y]:AXIS?

or set the trace type with:

DISP[~:A~ |:B|1|2][:Y]:AXIS< sp >< type >

where:

< type > ::= {GDELay|IMAGinary|LINMagnitude|LOGMagnitude|PHASe|REAL}

Table A-1 Normal Units For STAR, STOP, CENT and UNIT

	Trace Type: DISP:AXIS?			
Measurement TRAC:RES?	**LINM LOGM**	**PHAS**	**GDEL**	**REAL IMAG**
SPEC{1|2}	**V** V2 VRMS VRMS2 DBM DBVPK DBVRMS	**DEG** RAD	**S** MS‡‡ US‡‡	**V** V2 VRMS VRMS2
PSD{1|2}	**V/RTHZ** V2/HZ VRMS/RTHZ VRMS2/HZ DBVRMS/RTHZ DBVPK/RTHZ DBM/HZ	**DEG** RAD	**S** MS‡‡ US‡‡	**V/RTHZ** V2/HZ VRMS/RTHZ VRMS2/HZ
TIME{1|2}	**V** V2 VRMS VRMS2 DBM DBVPK DBVRMS	**DEG** RAD	–	**V** MV‡‡ UV‡‡
COH	**no units** DB	**DEG** RAD	**S** MS‡‡ US‡‡	**no units**
FRES	**no units** DB	**DEG** RAD	**S** MS‡‡ US‡‡	**no units**
CSP	**V2** VRMS2 DBVRMS DBVPK DBM	**DEG** RAD	**S** MS‡‡ US‡‡	**V2** VRMS2
F< num >*	**V** V2 VRMS VRMS2 DBM DBVPK DBVRMS	**DEG** RAD	**S** MS‡‡ US‡‡	**V** V2 VRMS VRMS2
F< num >† K< num >†	**no units**‡ DB††	**DEG** RAD	**S** MS‡‡ US‡‡	**no units**‡

** Use this row when the results of user math function are in V or V2.*
† Use this row when the results of user math function are NOT in V or V2.
‡ Uses the units of your math function or constant.
†† Converts your math units to a form of DB (precedes your math units with DB).
*‡‡ **NOT** valid for the DISP: Y: SCAL: UNITS command.*

Table A-2 Normal Units For DIV

	Trace Type: DISP:AXIS?				
Measurement TRAC:RES?	**LINM**	**LOGM**	**PHAS**	**GDEL**	**REAL IMAG**
SPEC{1 |2}	**V** V2 VRMS VRMS2	**DB**	**DEG** RAD	**S** MS US	**V** V2 VRMS VRMS2
PSD{1 |2}	**V/RTHZ** VRMS/RTHZ VRMS2/HZ V2/HZ	**DB**	**DEG** RAD	**S** MS US	**V/RTHZ** VRMS/RTHZ VRMS2/HZ V2/HZ
TIME{1 |2}	**V** V2 VRMS VRMS2	**DB**	**DEG** RAD	—	**V** MV UV
COH	**no units**	**DB**	**DEG** RAD	**S** MS US	**no units**
FRES	**no units**	**DB**	**DEG** RAD	**S** MS US	**no units**
CSP	**V2** VRMS2	**DB**	**DEG** RAD	**S** MS US	**V2** VRMS2
F< num >*	**V** V2 VRMS VRMS2	**DB**	**DEG** RAD	**S** MS US	**V** V2 VRMS VRMS2
F< num >† K< num >†	**no units**‡	**DB**	**DEG** RAD	**S** MS US	**no units**‡

** Use this row when the results of user math function are in V or V2.*
† Use this row when the results of user math function are NOT in V or V2.
‡ Uses the units of your math function or constant.

Table A-3 Engineering Units For STAR, STOP, CENT and UNIT

	Trace Type: DISP:AXIS?			
Measurement TRAC:RES?	**LINM LOGM**	**PHAS**	**GDEL**	**REAL IMAG**
SPEC{1|2}	**EU** EU2 EURMS EURMS2 DBEUPK DBEURMS	**DEG** RAD	**S** MS‡‡ US‡‡	**EU** EU2 EURMS EURMS2
PSD{1|2}	**EU/RTHZ** EU2/HZ EURMS/RTHZ EURMS2/HZ DBEURMS/RTHZ DBEUPK/RTHZ	**DEG** RAD	**S** MS‡‡ US‡‡	**EU/RTHZ** EU2/HZ EURMS/RTHZ EURMS2/HZ
TIME{1|2}	**EU** EU2 EURMS EURMS2 DBEUPK DBEURMS	**DEG** RAD	–	**EU** MEU‡‡ UEU‡‡
COH	**no units** DB	**DEG** RAD	**S** MS‡‡ US‡‡	**no units**
FRES	**no units** DB	**DEG** RAD	**S** MS‡‡ US‡‡	**no units**
CSP	**no units** DB	**DEG** RAD	**S** MS‡‡ US‡‡	**no units**
F< num >*	**EU** EU2 EURMS EURMS2 DBEUPK DBEURMS	**DEG** RAD	**S** MS‡‡ US‡‡	**EU** EU2 EURMS EURMS2
F< num >† K< num >†	**no units**‡ DB††	**DEG** RAD	**S** MS‡‡ US‡‡	**no units**‡

** Use this row when the results of user math function are in V or V2.*
† Use this row when the results of user math function are NOT in V or V2.
‡ Uses the units of your math function or constant.
†† Converts your math units to a form of DB (precedes your math units with DB).
‡‡ NOT valid for the DISP: Y: SCAL: UNITS command.

Table A-4 Engineering Units For DIV

	Trace Type: DISP:AXIS?				
Measurement TRAC:RES?	**LINM**	**LOGM**	**PHAS**	**GDEL**	**REAL IMAG**
SPEC{1 |2}	**EU** EU2 EURMS EURMS2	**DB**	**DEG** RAD	**S** MS US	**EU** EU2 EURMS EURMS2
PSD{1 |2}	**EU/RTHZ** EURMS/RTHZ EURMS2/HZ EU2/HZ	**DB**	**DEG** RAD	**S** MS US	**EU/RTHZ** EURMS/RTHZ EURMS2/HZ EU2/HZ
TIME{1 |2}	**EU** EU2 EURMS EURMS2	**DB**	**DEG** RAD	—	**EU** MEU UEU
COH	**no units**	**DB**	**DEG** RAD	**S** MS US	**no units**
FRES	**no units**††	**DB**	**DEG** RAD	**S** MS US	**no units**††
CSP	**no units**‡‡	**DB**	**DEG** RAD	**S** MS US	**no units**‡‡
F< num >*	**EU** EU2 EURMS EURMS2	**DB**	**DEG** RAD	**S** MS US	**EU** EU2 EURMS EURMS2
F< num >† K< num >†	**no units**‡	**DB**	**DEG** RAD	**S** MS US	**no units**‡

* *Use this row when the results of user math function are in V or V2.*
† *Use this row when the results of user math function are NOT inV or V2.*
‡ *Uses the units of your math function or constant.*
†† *Uses the units* $EU_2 \div EU_1$
‡‡ *Uses the units* $EU_1 \times EU_2$

Appendix B
Cross-Reference from Front-Panel Keys to HP-IB Commands

Introduction

This section lists analyzer hardkeys and softkeys and their equivalent HP-IB commands. Keys that do not have an equivalent HP-IB command are included to make the list of keys complete. The numbers beside each softkey indicate the position of the softkey on the screen—the top softkey is number 1 and the bottom softkey is number 10.

You can also determine the equivalent HP-IB command for any key sequence by turning mnemonic echo on. With mnemonic echo on, the analyzer displays the equivalent HP-IB command for each front panel key sequence. To turn mnemonic echo on, press the following keys:

<Local/HP-IB>
[HP-IB UTILITIES]
[MNEMONIC ECHO]

Measurement Group

FRONT PANEL KEY	HP-IB COMMAND
Average	
1) AVERAGE ON/OFF	AVER[:STAT] ON|OFF
2) NUMBER AVERAGES	AVER:COUN <NRf>
3) RMS AVERAGE	AVER:TYPE RMS;WEIG STAB
4) RMS EXPO AVERAGE	AVER:TYPE RMS;WEIG EXP
5) VECTOR AVERAGE	AVER:TYPE VECT;WEIG STAB
6) VECT EXPO AVERAGE	AVER:TYPE VECT;WEIG EXP
7) CONTINUOS PEAK HOLD	AVER:TYPE PEAK
8) FAST AVG ON/OFF	AVER:DISP:RATE:STAT ON|OFF
9) UPDATE RATE	AVER:DISP:RATE <NRf>
10) OVERLAP%	AVER:OVER <NRf> [PCT]
Frequency	
1) SPAN	FREQ:SPAN <NRf>
2) START	FREQ:STAR <NRf>
3) CENTER	FREQ:CENT <NRf>
4) ZERO START	FREQ:STAR 0
5) FULL SPAN	FREQ:SPAN:FULL
6) STEP	FREQ:CENT:STEP <NRf>
7) RECORD LENGTH	SWE:TIME <NRf>
Input	
1) CHANNEL 1 RANGE	INP1:RANG <NRf>
2) CHANNEL 1 AUTORANGE	INP1:RANG:AUTO ON
3) CHANNEL 1 SETUP	
1) FLOAT/GND	INP1:LOW FLO|GRO
2) AC/DC	INP1:COUP AC|DC
3) UNITS	
1) VOLTS	INP1:UNIT VOLT
2) ENG UNITS	INP1:UNIT EU
3) ENG UNIT VALUE	INP1:UNIT:EU:MULT <NRf>
4) ENG UNIT LABEL	INP1:UNIT:EU:NAME <string>
10) RETURN	
10) RETURN	
4) CHANNEL 2 RANGE	INP2:RANG <NRf>
5) CHANNEL 2 AUTORANGE	INP2:RANG:AUTO ON

FRONT PANEL KEY	HP-IB COMMAND
6) CHANNEL 2 SETUP	
1) FLOAT/GND	INP2:LOW FLO|GRO
2) AC/DC	INP2:COUP AC|DC
3) UNITS	
1) VOLTS	INP2:UNIT VOLT
2) ENG UNITS	INP2:UNIT EU
3) ENG UNIT VALUE	INP2:UNIT:EU:MULT <NRf>
4) ENG UNIT LABEL	INP2:UNIT:EU:NAME <string>
10) RETURN	
10) RETURN	
8) dBm REF IMPEDANCE	INP:IMP <NRf>
Meas Type	
1) 1 CHANNEL 102.4 KHZ	CONF:TYPE SPEC
2) 2 CHANNEL 51.2 KHZ	CONF:TYPE NETW
Pause/Cont	INIT:STAT PAUS|RUN
Source	
1) SOURCE ON/OFF	SOUR:STAT ON|OFF
2) LEVEL	SOUR:AMPL[:LEV] <NRf>
3) RANDOM	SOUR:FREQ:MODE RAND
4) PERIODIC CHIRP	SOUR:FREQ:MODE PCH
5) FIXED SINE	SOUR:FREQ:MODE CW
6) SINE FREQ ENTRY	SOUR:FREQ[:CW] <NRf>
Start	INIT:STAT STAR
Trigger	
1) CONTINUOS TRIGGER	TRIG:SOUR FREE
2) EXTERNAL TRIGGER	TRIG:SOUR EXT
3) CHANNEL 1 TRIGGER	TRIG:SOUR INT1
4) CHANNEL 2 TRIGGER	TRIG:SOUR INT2
5) SOURCE TRIGGER	TRIG:SOUR SOUR
6) HP-IB TRIGGER	TRIG:SOUR BUS
7) AUTOMATIC ARM	ARM:SOUR FREE
8) MANUAL ARM	ARM:SOUR HOLD
9) TRIGGER SET UP	
1) LEVEL	TRIG:LEV <NRf> [PCT]
2) SLOPE POS/NEG	TRIG:SLOP POS|NEG
3) CHANNEL 1 DELAY	TRIG:DEL1 <NRf>
4) CHANNEL 2 DELAY	TRIG:DEL2 <NRf>
10) RETURN	
10) ARM	ARM[:IMM]

FRONT PANEL KEY	HP-IB COMMAND
Window	
1) HANNING	WIND[:TYPE] HANN
2) FLAT TOP	WIND[:TYPE] FLAT
3) UNIFORM	WIND[:TYPE] UNIF
4) FORCE EXPO	WIND1 FORC;WIND2 EXP
6) FORCE CHANNEL 1	WIND1[:TYPE] FORC WIND1:CONS:FORC <NRf>
7) EXPO CHANNEL 1	WIND1[:TYPE] EXP WIND1:CONS:EXP <NRf>
9) FORCE CHANNEL 2	WIND2[:TYPE] FORC WIND2:CONS:FORC <NRf>
10) EXPO CHANNEL 2	WIND2[:TYPE] EXP WIND2:CONS:EXP <NRf>

Display Group

FRONT PANEL KEY	HP-IB COMMAND
Active Trace	SCR:ACT A|B
Format	————
1) SINGLE	SCR:FORM SING
2) UPPER/ LOWER	SCR:FORM ULOW
3) FRONT/ BACK	SCR:FORM FBAC
4) SETUP STATE	SCR:CONT STAT
5) TRCE GRID ON/OFF	DISP[:A|:B|1|2]:GRAT ON|OFF
6) TRACE TITLE	TRAC[:A|:B|1|2]:TITL <string>
9) DATA LBL ON/OFF	SCR:ANN ON|OFF
10) DISP BLNK ON/OFF	SCR[:STAT] OFF|ON
Math	————
1) DEFINE F1	USER:EXPR F1, <expression>
2) DEFINE F2	USER:EXPR F2, <expression>
3) DEFINE F3	USER:EXPR F3, <expression>
4) DEFINE F4	USER:EXPR F4, <expression>
5) DEFINE F5	USER:EXPR F5, <expression>
6) DEFINE K1	————
1) DEFINE REAL PART	USER:VAR:REAL K1, <NRf>
2) DEFINE IMAG PART	USER:VAR:IMAG K1, <NRf>
3) DEFINE MAGNITUDE	USER:VAR:MAGN K1, <NRf>
4) DEFINE PHASE	USER:VAR:PHAS K1, <NRf> [RAD|DEG]
10) RETURN	————
7) DEFINE K2	————
1) DEFINE REAL PART	USER:VAR:REAL K2, <NRf>
2) DEFINE IMAG PART	USER:VAR:IMAG K2, <NRf>
3) DEFINE MAGNITUDE	USER:VAR:MAGN K2, <NRf>
4) DEFINE PHASE	USER:VAR:PHAS K2, <NRf> [RAD|DEG]
10) RETURN	————
8) DEFINE K3	
1) DEFINE REAL PART	USER:VAR:REAL K3, <NRf>
2) DEFINE IMAG PART	USER:VAR:IMAG K3, <NRf>
3) DEFINE MAGNITUDE	USER:VAR:MAGN K3, <NRf>
4) DEFINE PHASE	USER:VAR:PHAS K3, <NRf> [RAD|DEG]
10) RETURN	————

FRONT PANEL KEY	HP-IB COMMAND
9) DEFINE K4	
1) DEFINE REAL PART	USER:VAR:REAL K4, <NRf>
2) DEFINE IMAG PART	USER:VAR:IMAG K4, <NRf>
3) DEFINE MAGNITUDE	USER:VAR:MAGN K4, <NRf>
4) DEFINE PHASE	USER:VAR:PHAS K4, <NRf> [RAD|DEG]
10) RETURN	
10) DEFINE K5	
1) DEFINE REAL PART	USER:VAR:REAL K5, <NRf>
2) DEFINE IMAG PART	USER:VAR:IMAG K5, <NRf>
3) DEFINE MAGNITUDE	USER:VAR:MAGN K5, <NRf>
4) DEFINE PHASE	USER:VAR:PHAS K5, <NRf> [RAD|DEG]
10) RETURN	
Meas Data	
1) SPECTRUM CHANNEL 1	TRAC[:A|:B|1|2]:RES SPEC1
2) SPECTRUM CHANNEL 2	TRAC[:A|:B|1|2]:RES SPEC2
3) PSD CHANNEL 1	TRAC[:A|:B|1|2]:RES PSD1
4) PSD CHANNEL 2	TRAC[:A|:B|1|2]:RES PSD2
5) TIME CHANNEL 1	TRAC[:A|:B|1|2]:RES TIME1
6) TIME CHANNEL 2	TRAC[:A|:B|1|2]:RES TIME2
7) FREQUENCY RESPONSE	TRAC[:A|:B|1|2]:RES FRES
8) COHERENCE	TRAC[:A|:B|1|2]:RES COH
9) CROSS SPECTRUM	TRAC[:A|:B|1|2]:RES CSP
10) MORE	
1) FUNCTION (F1-F5)	
1) FUNCTION F1	TRAC[:A|:B|1|2]:RES F1
2) FUNCTION F2	TRAC[:A|:B|1|2]:RES F2
3) FUNCTION F3	TRAC[:A|:B|1|2]:RES F3
4) FUNCTION F4	TRAC[:A|:B|1|2]:RES F4
5) FUNCTION F5	TRAC[:A|:B|1|2]:RES F5
10) RETURN	
2) CONSTANT (K1-K5)	
1) CONSTANT K1	TRAC[:A|:B|1|2]:RES K1
2) CONSTANT K2	TRAC[:A|:B|1|2]:RES K2
3) CONSTANT K3	TRAC[:A|:B|1|2]:RES K3
4) CONSTANT K4	TRAC[:A|:B|1|2]:RES K4
5) CONSTANT K5	TRAC[:A|:B|1|2]:RES K5
10) RETURN	

FRONT PANEL KEY	HP-IB COMMAND
3) RECALL TRACE	
1) RCL FROM 'TRACE1'	MMEM:GET|LOAD:TRAC[:A|:B|1|2] "TRACE1"
2) RCL FROM 'TRACE2'	MMEM:GET|LOAD:TRAC[:A|:B|1|2] "TRACE2"
3) RCL FROM 'TRACE3'	MMEM:GET|LOAD:TRAC[:A|:B|1|2] "TRACE3"
4) RCL FROM 'TRACE4'	MMEM:GET|LOAD:TRAC[:A|:B|1|2] "TRACE4"
5) RCL FROM 'TRACE5'	MMEM:GET|LOAD:TRAC[:A|:B|1|2] "TRACE5"
6) RCL FROM 'TRACE6'	MMEM:GET|LOAD:TRAC[:A|:B|1|2] "TRACE6"
7) RCL FROM 'TRACE7'	MMEM:GET|LOAD:TRAC[:A|:B|1|2] "TRACE7"
8) RCL FROM 'TRACE8'	MMEM:GET|LOAD:TRAC[:A|:B|1|2] "TRACE8"
9) DEFINE FILENAME	MMEM:GET|LOAD:TRAC[:A|:B|1|2] "<filename>"
10) RETURN	———
10) RETURN	———
Trace Type	———
1) LINEAR MAGNITUDE	DISP[:A|:B|1|2]:AXIS LINM
2) LOG MAGNITUDE	DISP[:A|:B|1|2]:AXIS LOGM
3) PHASE	DISP[:A|:B|1|2]:AXIS PHAS
4) GROUP DELAY	DISP[:A|:B|1|2]:AXIS GDEL
1) APERT .5% OF SPAN	DISP[:A|:B|1|2]:X:APER 0.5 PCT
2) APERT 1% OF SPAN	DISP[:A|:B|1|2]:X:APER 1 PCT
3) APERT 2% OF SPAN	DISP[:A|:B|1|2]:X:APER 2 PCT
4) APERT 4% OF SPAN	DISP[:A|:B|1|2]:X:APER 4 PCT
5) APERT 8% OF SPAN	DISP[:A|:B|1|2]:X:APER 8 PCT
6) APERT 16% OF SPAN	DISP[:A|:B|1|2]:X:APER 16 PCT
10) RETURN	———
5) REAL PART	DISP[:A|:B|1|2]:AXIS REAL
6) IMAGINARY PART	DISP[:A|:B|1|2]:AXIS IMAG
Scale	———
1) AUTO SCALE	DISP[:A|:B|1|2][:Y]:SCAL:AUTO:SING
2) TOP REFERENCE	DISP[:A|:B|1|2][:Y]:SCAL:STOP <NRf>
3) CENTER REFERENCE	DISP[:A|:B|1|2][:Y]:SCAL:CENT <NRf>
4) BOTTOM REFERENCE	DISP[:A|:B|1|2][:Y]:SCAL:STAR <NRf>
5) REF LEVEL TRACKING	DISP[:A|:B|1|2][:Y]:SCAL:REF INP
6) VERTICAL /DIV	DISP[:A|:B|1|2][:Y]:SCAL:DIV <NRf>
7) VERTICAL UNITS	DISP[:A|:B|1|2][:Y]:SCAL:UNIT <string>
8) X-AXIS LIN/LOG	DISP[:A|:B|1|2]:X:SPAC LIN|LOG

Marker Group

FRONT PANEL KEY	HP-IB COMMAND
↑	MARK[:A|:B|1|2]:AMAX:RIGH
↓	MARK[:A|:B|1|2]:AMAX:LEFT
←	MARK[:A|:B|1|2]:POIN DOWN
→	MARK[:A|:B|1|2]:POIN UP
Marker	
1) MARKER ON/OFF	MARK[:A|:B|1|2][:X]:STAT ON|OFF
2) COUPLED ON/OFF	MARK[:A|:B|1|2][:X]:AUTO ON|OFF
3) X ENTRY	MARK[:A|:B|1|2][:X]: <NRf>
4) OFFSET	
1) OFFSET ON/OFF	MARK[:A|:B|1|2][:X]:MODE DELT|NORM
2) OFFSET ZERO	MARK[:A|:B|1|2][:X]:DELT:ZERO
3) REFERENCE X ENTRY	MARK[:A|:B|1|2][:X]:DELT <NRf>
4) REFERENCE Y ENTRY	MARK[:A|:B|1|2][:X]:DELT:AMPL <NRf>
10) RETURN	
5) MARKER TO PEAK	MARK[:A|:B|1|2][:X]:AMAX[:GLOB]
6) NXT RIGHT PEAK	MARK[:A|:B|1|2][:X]:AMAX:RIGH
7) NXT LEFT PEAK	MARK[:A|:B|1|2][:X]:AMAX:LEFT
8) MARKER TO MINIMUM	MARK[:A|:B|1|2][:X]:AMIN[:GLOB]
9) PEAK TRK ON/OFF	MARK[:A|:B|1|2][:X]:AMAX:AUTO ON|OFF
10) SEARCH	
1) TARGET	MARK[:A|:B|1|2][:X]:SEAR:AMPL <NRf>
2) LEFT	MARK[:A|:B|1|2][:X]:SEAR:LEFT
3) RIGHT	MARK[:A|:B|1|2][:X]:SEAR:RIGH
10) RETURN	
Marker Fctn	
1) OFF	MARK[:A|:B|1|2]:FUNC AOFF
2) HARMONIC	MARK[:A|:B|1|2]:HARM:STAT ON
1) FNDMNTL FREQ	MARK[:A|:B|1|2]:HARM[:FREQ] <NRf>
2) DEFINE NUM HARM	MARK[:A|:B|1|2]:HARM:COUN <NRf>
3) DIVIDE FNDMNTL	MARK[:A|:B|1|2]:HARM:FREQ:DIV <NRf>
4) THD	MARK[:A|:B|1|2]:HARM:THD:STAT ON
5) HARM PWR	MARK[:A|:B|1|2]:HARM:POW:STAT ON
9) RESULTS ON/OFF	MARK[:A|:B|1|2]:HARM:POW:STAT ON|OFF
10) RETURN	

FRONT PANEL KEY	HP-IB COMMAND
3) SIDEBAND	MARK[:A|:B|1|2]:SID:STAT ON
1) CARRIER FREQ	MARK[:A|:B|1|2]:SID[:FREQ] <NRf>
2) SIDEBAND INCREM	MARK[:A|:B|1|2]:SID:DELT <NRf>
3) DEFINE NUM SDBND	MARK[:A|:B|1|2]:SID:COUN <NRf>
9) RESULTS ON/OFF	MARK[:A|:B|1|2]:SID:POW:STAT ON|OFF
10) RETURN	_____
4) BAND	MARK[:A|:B|1|2]:BAND:STAT ON
1) DEFINE LEFT FREQ	MARK[:A|:B|1|2]:BAND:STAR <NRf>
2) DEFINE RGHT FREQ	MARK[:A|:B|1|2]:BAND:STOP <NRf>
3) DEFINE CENT FREQ	MARK[:A|:B|1|2]:BAND:CENT <NRf>
9) RESULTS ON/OFF	MARK[:A|:B|1|2]:BAND:POW:STAT ON|OFF
10) RETURN	_____
5) LIMIT	_____
1) X-START	_____
2) Y-START	_____
3) X-STOP	_____
4) Y-STOP	_____
5) LIMIT UPPER/LOW	_____
6) INSERT SEGMENT	_____
7) DELETE	_____
1) DELETE SEGMENT	_____
4) DELETE ALL	_____
10) RETURN	_____
8) OFFSET	_____
1) Y OFFSET ON/OFF	_____
2) Y OFFSET VALUE	_____
4) X ADJUST ALL SEGS	_____
5) Y ADJUST ALL SEGS	_____
10) RETURN	_____
9) LIMIT CONFIG	_____
1) LINES ON/OFF	DISP[:A|:B|1|2]:LIM:LINE ON|OFF
2) TEST EVAL ON/OFF	DISP[:A|:B|1|2]:LIM:STAT ON|OFF
3) BEEP ON/OFF	DISP[:A|:B|1|2]:LIM:BEEP ON|OFF

FRONT PANEL KEY	HP-IB COMMAND
5) SELECT LIMIT	————
1) L1 ACTIVE	DISP[:A|:B|1|2]:LIM[:TABL] 1
2) L2 ACTIVE	DISP[:A|:B|1|2]:LIM[:TABL] 2
3) L3 ACTIVE	DISP[:A|:B|1|2]:LIM[:TABL] 3
4) L4 ACTIVE	DISP[:A|:B|1|2]:LIM[:TABL] 4
5) L5 ACTIVE	DISP[:A|:B|1|2]:LIM[:TABL] 5
6) L6 ACTIVE	DISP[:A|:B|1|2]:LIM[:TABL] 6
7) L7 ACTIVE	DISP[:A|:B|1|2]:LIM[:TABL] 7
8) L8 ACTIVE	DISP[:A|:B|1|2]:LIM[:TABL] 8
10) RETURN	————
6) SELECT SEGMENT	————
10) RETURN	————
10) RETURN TBL DOWN	————
6) DATA TABLE	————
1) CALC ON/OFF	MARK[:A|:B|1|2]:DTAB:STAT ON|OFF
2) EDIT X	————
3) INSERT X	————
4) MOVE TO ENTRY NUM	————
6) DELETE ENTRY	————
7) DELETE ALL	————
1) DO DELETE	————
10) RETURN	————
10) RETURN TBL DOWN	————

System Group

FRONT PANEL KEY	HP-IB COMMAND
Help	————
Local/HP-IB	————
1) SYSTEM CONTROLLR	————
2) ADDRESSBL ONLY	————
3) ANALYZER ADDRESS	SYST:ADDR <NRf>
4) PERIPHERL ADDRESSES	————
1) DISC ADDRESS	MMEM:MSI:ADDR <NRf>
2) DISC UNIT	MMEM:MSI:UNIT <NRf>
3) DISC VOLUME	MMEM:MSI:VOL <NRf>
4) PLOTTER ADDRESS	PLOT:ADDR <NRf>
5) PRINTER ADDRESS	PRIN:ADDR <NRf>
10) RETURN	————
5) HP-IB UTILITIES	————
1) STATUS ON/OFF	GPIB:LEDS ON|OFF
2) MNEMONIC OFF	GPIB:MNEM OFF
3) MNEMONIC ECHO	GPIB:MNEM ECHO
4) HP-IB SCROLL	GPIB:MNEM SCR
10) RETURN	————
10) USER SRQ	————
1) USER SRQ 0	STAT:USER:PULS 1
2) USER SRQ 1	STAT:USER:PULS 2
3) USER SRQ 2	STAT:USER:PULS 4
4) USER SRQ 3	STAT:USER:PULS 8
5) USER SRQ 4	STAT:USER:PULS 16
6) USER SRQ 5	STAT:USER:PULS 32
7) USER SRQ 6	STAT:USER:PULS 64
8) USER SRQ 7	STAT:USER:PULS 128
9) USER SRQ 8	STAT:USER:PULS 256
10) USER SRQ 9	STAT:USER:PULS 512
Plot/Print	————
1) PLOT SCREEN	PLOT:DUMP:SCR
1) ABORT PLOT	————
2) PLOT TRACE	PLOT:DUMP:TRAC
1) ABORT PLOT	————
3) PLOT MARKER	PLOT:DUMP:MARK
1) ABORT PLOT	————

FRONT PANEL KEY	HP-IB COMMAND
4) DEFINE PLT SPEED	————
1) SLOW (5 cm/s)	PLOT:SPE 5
2) FAST (36 cm/s)	PLOT:SPE 36
3) USER DEFINED	PLOT:SPE <NRf>
4) USER ENTRY	————
10) RETURN	————
5) DEFINE PLOT PENS	————
1) DEFAULT PENS	PLOT:PEN:INIT
2) TRACE A PEN NUM	PLOT:PEN:TRAC:A <NRf>
3) TRACE A LINE TYPE	————
1) SOLID	PLOT:LTYP:TRAC:A -4096
2) DOTTED	PLOT:LTYP:TRAC:A 1
3) DASHED	PLOT:LTYP:TRAC:A 2
4) USER DEFINED	PLOT:LTYP:TRAC:A <NRf>
5) USER TYPE ENTRY	————
10) RETURN	————
5) TRACE B PEN NUM	PLOT:PEN:TRAC:B <NRf>
6) TRACE B LINE TYPE	————
1) SOLID	PLOT:LTYP:TRAC:B -4096
2) DOTTED	PLOT:LTYP:TRAC:B 1
3) DASHED	PLOT:LTYP:TRAC:B 2
4) USER DEFINED	PLOT:LTYP:TRAC:B <NRf>
5) USER TYPE ENTRY	————
10) RETURN	————
8) ALPHA PEN NUM	PLOT:PEN:ALPH <NRf>
9) GRID PEN NUM	PLOT:PEN:GRID <NRf>
10) RETURN	————
7) PRINT SCREEN	PRIN:DUMP:SCR
1) ABORT PRINT	————
8) PRINT ALPHA	PRIN:DUMP:ALPH
1) ABORT PRINT	————
9) PERIPHERL ADDRESSES	————
1) DISC ADDRESS	MMEM:MSI:ADDR <NRf>
2) DISC UNIT	MMEM:MSI:UNIT <NRf>
3) DISC VOLUME	MMEM:MSI:VOL <NRf>
4) PLOTTER ADDRESS	PLOT:ADDR <NRf>
5) PRINTER ADDRESS	PRIN:ADDR <NRf>
10) RETURN	————

FRONT PANEL KEY	HP-IB COMMAND
Preset	*RST
Save	_____
1) SAVE TRACE	_____
1) INTO FILE 'TRACE1'	MMEM:SAVE|STOR:TRAC[:A|:B|1|2] "TRACE1"
2) INTO FILE 'TRACE2'	MMEM:SAVE|STOR:TRAC[:A|:B|1|2] "TRACE2"
3) INTO FILE 'TRACE3'	MMEM:SAVE|STOR:TRAC[:A|:B|1|2] "TRACE3"
4) INTO FILE 'TRACE4'	MMEM:SAVE|STOR:TRAC[:A|:B|1|2] "TRACE4"
5) INTO FILE 'TRACE5'	MMEM:SAVE|STOR:TRAC[:A|:B|1|2] "TRACE5"
6) INTO FILE 'TRACE6'	MMEM:SAVE|STOR:TRAC[:A|:B|1|2] "TRACE6"
7) INTO FILE 'TRACE7'	MMEM:SAVE|STOR:TRAC[:A|:B|1|2] "TRACE7"
8) INTO FILE 'TRACE8'	MMEM:SAVE|STOR:TRAC[:A|:B|1|2] "TRACE8"
9) DEFINE FILENAME	MMEM:SAVE|STOR:TRAC[:A|:B|1|2] "<filename>"
10) RETURN	_____
2) SAVE STATE	_____
1) INTO FILE 'STATE1'	MMEM:SAVE|STOR:STAT "STATE1"
2) INTO FILE 'STATE2'	MMEM:SAVE|STOR:STAT "STATE2"
3) INTO FILE 'STATE3'	MMEM:SAVE|STOR:STAT "STATE3"
4) INTO FILE 'STATE4'	MMEM:SAVE|STOR:STAT "STATE4"
5) INTO FILE 'STATE5'	MMEM:SAVE|STOR:STAT "STATE5"
6) INTO FILE 'STATE6'	MMEM:SAVE|STOR:STAT "STATE6"
7) INTO FILE 'STATE7'	MMEM:SAVE|STOR:STAT "STATE7"
8) INTO FILE 'STATE8'	MMEM:SAVE|STOR:STAT "STATE8"
9) DEFINE FILENAME	MMEM:SAVE|STOR:STAT "<filename>"
10) RETURN	_____
3) SAVE MATH	_____
1) INTO FILE 'MATH1'	MMEM:SAVE|STOR:MATH "MATH1"
2) INTO FILE 'MATH2'	MMEM:SAVE|STOR:MATH "MATH2"
3) INTO FILE 'MATH3'	MMEM:SAVE|STOR:MATH "MATH3"
4) INTO FILE 'MATH4'	MMEM:SAVE|STOR:MATH "MATH4"
5) INTO FILE 'MATH5'	MMEM:SAVE|STOR:MATH "MATH5"
6) INTO FILE 'MATH6'	MMEM:SAVE|STOR:MATH "MATH6"
7) INTO FILE 'MATH7'	MMEM:SAVE|STOR:MATH "MATH7"
8) INTO FILE 'MATH8'	MMEM:SAVE|STOR:MATH "MATH8"
9) DEFINE FILENAME	MMEM:SAVE|STOR:MATH "<filename>"
10) RETURN	_____
4) SAVE MORE	_____
1) SAVE LIMIT	MMEM:SAVE|STOR:LIM<limit#>" <filename>"
2) SAVE DATA TABLE	MMEM:SAVE|STOR:DTAB[:A|:B|1|2]"< filename>"
10) RETURN	_____

FRONT PANEL KEY	HP-IB COMMAND
6) SAVE SYS CONFIG	————
1) DO SAVE	SYST:SAVE
10) CANCEL/RETURN	————
7) FILE UTILITIES	————
1) RENAME FILE	MMEM:REN" <old filename>","<new filename>"
2) DELETE FILE	MMEM:DEL" <filename>"
3) DELETE ALL FILES	MMEM:DEL" <msi>"
4) PACK FILES	MMEM:PACK "<msi>"
5) RENAME CATALOG	MMEM:REN" <msi>","<new cat name>"
6) COPY DISC	————
1) SOURCE DISC	————
2) DESTN DISC	————
4) START COPY	MMEM:COPY "<source device>","<destn filename>"
10) RETURN	————
7) COPY FILE	————
1) SOURCE FILENAME	————
2) DESTN FILENAME	————
4) START COPY	MMEM:COPY "<source device>","<destn filename>"
10) RETURN	————
8) STORAGE CONFIG	————
1) INTERNAL RAM DISC	MMEM:MSI "RAM:"
2) INTERNAL DISC	MMEM:MSI "INT:"
3) EXTERNAL DISC	MMEM:MSI "EXT:"
7) FORMAT ASCII/BIN	MMEM:FORM ASC|BIN
8) PERIPHERL ADDRESSES	————
1) DISC ADDRESS	MMEM:MSI:ADDR <NRf>
2) DISC UNIT	MMEM:MSI:UNIT <NRf>
3) DISC VOLUME	MMEM:MSI:VOL <NRf>
4) PLOTTER ADDRESS	PLOT:ADDR <NRf>
5) PRINTER ADDRESS	PRIN:ADDR <NRf>
10) RETURN	————
9) CATALOG ON/OFF	SCR:CONT DCAT
10) RETURN	————
9) DISC FUNCTIONS	————
1) FORMAT OPTION	MMEM:INIT:OPT <NRf>
2) INTRLEAVE FACTOR	MMEM:INIT:INT <NRf>
4) START FORMAT	MMEM:INIT "<msi>"

FRONT PANEL KEY	HP-IB COMMAND
8) STORAGE CONFIG	————
1) INTERNAL RAM DISC	MMEM:MSI "RAM:"
2) INTERNAL DISC	MMEM:MSI "INT:"
3) EXTERNAL DISC	MMEM:MSI "EXT:"
7) FORMAT ASCII/BIN	MMEM:FORM ASC|BIN
8) PERIPHERL ADDRESSES	————
1) DISC ADDRESS	MMEM:MSI:ADDR <NRf>
2) DISC UNIT	MMEM:MSI:UNIT <NRf>
3) DISC VOLUME	MMEM:MSI:VOL <NRf>
4) PLOTTER ADDRESS	PLOT:ADDR <NRf>
5) PRINTER ADDRESS	PRIN:ADDR <NRf>
10) RETURN	————
9) CATALOG ON/OFF	SCR:CONT DCAT
10) RETURN	————
9) CATALOG ON/OFF	SCR:CONT DCAT
10) RETURN	————
10) RETURN	————
8) STORAGE CONFIG	————
1) INTERNAL RAM DISC	MMEM:MSI "RAM:"
2) INTERNAL DISC	MMEM:MSI "INT:"
3) EXTERNAL DISC	MMEM:MSI "EXT:"
7) FORMAT ASCII/BIN	MMEM:FORM ASC|BIN
8) PERIPHERL ADDRESSES	————
1) DISC ADDRESS	MMEM:MSI:ADDR <NRf>
2) DISC UNIT	MMEM:MSI:UNIT <NRf>
3) DISC VOLUME	MMEM:MSI:VOL <NRf>
4) PLOTTER ADDRESS	PLOT:ADDR <NRf>
5) PRINTER ADDRESS	PRIN:ADDR <NRf>
10) RETURN	————
9) CATALOG ON/OFF	SCR:CONT DCAT
10) RETURN	————
9) CATALOG ON/OFF	SCR:CONT DCAT
10) DISC FUNCTIONS	————
1) FORMAT OPTION	MMEM:INIT:OPT <NRf>
2) INTRLEAVE FACTOR	MMEM:INIT:INT <NRf>
4) START FORMAT	MMEM:INIT "<msi>"

FRONT PANEL KEY	HP-IB COMMAND
8) STORAGE CONFIG	————
1) INTERNAL RAM DISC	MMEM:MSI "RAM:"
2) INTERNAL DISC	MMEM:MSI "INT:"
3) EXTERNAL DISC	MMEM:MSI "EXT:"
7) FORMAT ASCII/BIN	MMEM:FORM ASC|BIN
8) PERIPHERL ADDRESSES	————
1) DISC ADDRESS	MMEM:MSI:ADDR <NRf>
2) DISC UNIT	MMEM:MSI:UNIT <NRf>
3) DISC VOLUME	MMEM:MSI:VOL <NRf>
4) PLOTTER ADDRESS	PLOT:ADDR <NRf>
5) PRINTER ADDRESS	PRIN:ADDR <NRf>
9) CATALOG ON/OFF	SCR:CONT DCAT
10) RETURN	————
9) CATALOG ON/OFF	SCR:CONT DCAT
10) DISC FUNCTIONS	————
Spcl Fctn	————
1) SINGLE CAL	CAL:SING
2) AUTO CAL ON/OFF	CAL:AUTO ON|OFF
3) CAL OPTIONS	————
1) CLEAR CAL CONSTANTS	CAL:CLE
2) CAL TRACE ON/OFF	CAL:TRAC ON|OFF
10) RETURN	————
4) BEEPER ON/OFF	SYST:BEEP ON|OFF
5) TIME HHMMSS	SYST:TIME <hour>,<min>,<sec>
6) DATE MMDD[YY]	SYST:DATE <year>,<mon>,<day>
7) FAULT LOG	SCR:CONT FLOG
1) CLEAR FAULT LOG	SYST:FLOG:CLE
2) DESCRIBE ENTRY	SCR:CONT FLOG;:SYST:FLOG:ENTR:DESC <NRf>
8) VERSION	SYST:VERS
10) RETURN	————
8) MEMORY USAGE	SCR:CONT MEM
10) RETURN	————
9) SELF TEST	————
1) QUICK CONF TEST	TEST:SHOR
2) LONG CONF TEST	————
1) START	TEST:LCON
10) ABORT RETURN	TEST:ABOR
3) FUNCTIONL TESTS	————

FRONT PANEL KEY	HP-IB COMMAND
1) CPU ROM/RAM	————
1) CPU	TEST:PROC:CPU
2) ROM	TEST:PROC:ROM
3) RAM	TEST:PROC:RAM
4) INTERRUPT	TEST:PROC:INT
5) MULT FCTN PERIPHERL	TEST:PROC:MFP
8) ALL	TEST:PROC:ALL
10) ABORT RETURN	TEST:ABOR
2) DISPLAY	————
1) TEST PATTERN	TEST:DISP:PATT ON
10) ABORT RETURN	TEST:DISP:PATT OFF
2) DISPLAY: TEST	TEST:DISP
10) ABORT RETURN	TEST:ABOR
3) DMA	TEST:DMA
4) I/O	————
1) KEYBOARD	TEST:IO:KEY
2) HP-IB	————
1) HP-IB FUNC TEST	————
2) HP-IB CONNECTOR	————
10) ABORT RETURN	————
3) INTERNAL DISC	————
1) DISC CONTROLLR	TEST:IO:DISC:CONT
2) MOTOR	TEST:IO:DISC:MOT
3) RESTORE	TEST:IO:DISC:REST
4) RANDOM SEEK	TEST:IO:DISC:RAND
5) SEEK SECTOR	TEST:IO:DISC:SEEK <NRf>
6) READ	TEST:IO:DISC:READ
7) READ/ WRITE	TEST:IO:DISC:WRIT
8) READ/ WRITE ALL	TEST:IO:DISC:RW
9) ALL	TEST:IO:DISC:ALL
10) ABORT RETURN	TEST:ABOR

FRONT PANEL KEY	HP-IB COMMAND
4) IIC BUS	TEST:IO:SBUS
5) FAST BUS	TEST:IO:FBUS
8) ALL	TEST:IO:ALL
10) ABORT RETURN	TEST:ABOR
5) MATH COPROCSSR	TEST:MATH
6) DSP	————
1) TRIGGER	TEST:DSP:TRIG
2) LO	TEST:DSP:LO
3) DIGITAL FILTER	TEST:DSP:FILT
4) FIFO	TEST:DSP:FIFO
5) BASEBAND	TEST:DSP:BASE
6) ZOOM	TEST:DSP:ZOOM
7) DGTL SRCE THRU DSP	TEST:DSP:SOUR
8) ALL	TEST:DSP:ALL
10) ABORT RETURN	TEST:ABOR
7) SOURCE	————
1) SOURCE LO	TEST:SOUR:LO
2) SOURCE TO CPU	TEST:SOUR:CPU
3) WITHOUT LO	TEST:SOUR:BASE
4) WITH LO	TEST:SOUR:ZOOM
8) ALL	TEST:SOUR:ALL
10) ABORT RETURN	TEST:ABOR
8) OTHER	————
1) ADC GATE ARRAY	TEST:REC:GARR
2) INPUTS	TEST:REC:INP
10) ABORT RETURN	TEST:ABOR
9) ALL	TEST:ALL
10) ABORT RETURN	TEST:ABOR
4) LOOP MODE	————
1) LOOP MODE ON/OFF	TEST:LOOP[:STAT] ON|OFF
2) LOOP ALL	TEST:LOOP:STAR
10) ABORT LOOP TEST	TEST:ABOR
5) TEST LOG	SCR:CONT TLOG
1) CLEAR TEST LOG	TEST:LOG:CLE
2) TEST LOG OFF	SCR:CONT TRAC
10) RETURN	————
10) ABORT RETURN	TEST:ABOR

FRONT PANEL KEY	HP-IB COMMAND
10) SERVICE TESTS	————
1) ADJUSTMTS	————
1) CHANNEL 1 OFFSET	SERV:ADJ:OFFS1
2) CHANNEL 1 CMRR	SERV:ADJ:CMRR1
3) CHANNEL 1 FLATNESS	————
1) 50 kHz	SERV:ADJ:FLAT1:CENT
2) 100 kHz	SERV:ADJ:FLAT1:FULL
10) RETURN	————
4) CHANNEL 2 OFFSET	SERV:ADJ:OFFS2
5) CHANNEL 2 CMRR	SERV:ADJ:CMRR2
6) CHANNEL 2 FLATNESS	SERV:ADJ:FLAT2
7) ADC	————
1) SECOND PASS GAIN	SERV:ADJ:ADC:GAIN
2) OFFSET	SERV:ADJ:ADC:OFFS
10) RETURN	————
10) RETURN	————
2) SPCL TEST MODES	————
1) DITHER ON/OFF	SERV:DITH ON|OFF
2) TRK ALWAYS ON/OFF	SERV:TRAC ON|OFF
4) INPUTS: FRONT BNC	————
5) HIGH LEVEL CAL	SERV:CAL:HIGH ON
6) LOW LEVEL CAL	SERV:CAL:LOW ON
7) SOURCE LEVEL	SERV:SOUR:INP ON
10) RETURN	————
8) SERIAL: NUMBER	————
10) RETURN	————
10) RETURN	————
Recall	————
1) RECALL TRACE	————
1) RCL FROM 'TRACE1'	MMEM:GET|LOAD:TRAC[:A|:B|1|2] "TRACE1"
2) RCL FROM 'TRACE2'	MMEM:GET|LOAD:TRAC[:A|:B|1|2] "TRACE2"
3) RCL FROM 'TRACE3'	MMEM:GET|LOAD:TRAC[:A|:B|1|2] "TRACE3"
4) RCL FROM 'TRACE4'	MMEM:GET|LOAD:TRAC[:A|:B|1|2] "TRACE4"
5) RCL FROM 'TRACE5'	MMEM:GET|LOAD:TRAC[:A|:B|1|2] "TRACE5"
6) RCL FROM 'TRACE6'	MMEM:GET|LOAD:TRAC[:A|:B|1|2] "TRACE6"
7) RCL FROM 'TRACE7'	MMEM:GET|LOAD:TRAC[:A|:B|1|2] "TRACE7"
8) RCL FROM 'TRACE8'	MMEM:GET|LOAD:TRAC[:A|:B|1|2] "TRACE8"
9) DEFINE FILENAME	MMEM:GET|LOAD:TRAC[:A|:B|1|2] "<filename>"
10) RETURN	————

FRONT PANEL KEY	HP-IB COMMAND
2) RECALL STATE	
1) RCL FROM 'STATE1'	MMEM:GET|LOAD:STAT "STATE1"
2) RCL FROM 'STATE2'	MMEM:GET|LOAD:STAT "STATE2"
3) RCL FROM 'STATE3'	MMEM:GET|LOAD:STAT "STATE3"
4) RCL FROM 'STATE4'	MMEM:GET|LOAD:STAT "STATE4"
5) RCL FROM 'STATE5'	MMEM:GET|LOAD:STAT "STATE5"
6) RCL FROM 'STATE6'	MMEM:GET|LOAD:STAT "STATE6"
7) RCL FROM 'STATE7'	MMEM:GET|LOAD:STAT "STATE7"
8) RCL FROM 'STATE8'	MMEM:GET|LOAD:STAT "STATE8"
9) DEFINE FILENAME	MMEM:GET|LOAD:STAT "<filename>"
10) RETURN	
3) RECALL MATH	
1) RCL FROM 'MATH1'	MMEM:GET|LOAD:MATH "MATH1"
2) RCL FROM 'MATH2'	MMEM:GET|LOAD:MATH "MATH2"
3) RCL FROM 'MATH3'	MMEM:GET|LOAD:MATH "MATH3"
4) RCL FROM 'MATH4'	MMEM:GET|LOAD:MATH "MATH4"
5) RCL FROM 'MATH5'	MMEM:GET|LOAD:MATH "MATH5"
6) RCL FROM 'MATH6'	MMEM:GET|LOAD:MATH "MATH6"
7) RCL FROM 'MATH7'	MMEM:GET|LOAD:MATH "MATH7"
8) RCL FROM 'MATH8'	MMEM:GET|LOAD:MATH "MATH8"
9) DEFINE FILENAME	MMEM:GET|LOAD:MATH" <filename>"
10) RETURN	
4) RECALL MORE	
1) RECALL LIMIT	MMEM:GET|LOAD:LIM<limit#>" <filename>"
2) RECALL DATA TABL	MMEM:GET|LOAD:DTAB<:A|:B|1|2> "<filename>"
10) RETURN	
5) APPLICATN UTILITIES	
1) LIST ON/OFF	SCR:CONT APPL
2) LOAD APPLICATN	MMEM:LOAD:APPL "<filename>"
3) LOAD ALL	MMEM:LOAD:APPL:ALL" <msi>"
4) AUTO LOAD ON/OFF	MMEM:LOAD:APPL:AUTO ON|OFF
10) RETURN	
7) FILE UTILITIES (SEE **Save** - FILE UTILITIES)	

FRONT PANEL KEY	HP-IB COMMAND
8) STORAGE CONFIG	————
1) INTERNAL RAM DISC	MMEM:MSI "RAM:"
2) INTERNAL DISC	MMEM:MSI "INT:"
3) EXTERNAL DISC	MMEM:MSI "EXT:"
7) FORMAT ASCII/BIN	MMEM:FORM ASC|BIN
8) PERIPHERL ADDRESSES	————
1) DISC ADDRESS	MMEM:MSI:ADDR <NRf>
2) DISC UNIT	MMEM:MSI:UNIT <NRf>
3) DISC VOLUME	MMEM:MSI:VOL <NRf>
4) PLOTTER ADDRESS	PLOT:ADDR <NRf>
5) PRINTER ADDRESS	PRIN:ADDR <NRf>
10) RETURN	————
9) CATALOG ON/OFF	SCR:CONT DCAT
10) RETURN	————
9) CATALOG ON/OFF	SCR:CONT DCAT
10) DISC FUNCTIONS	————
1) FORMAT OPTION	MMEM:INIT:OPT <NRf>
2) INTRLEAVE FACTOR	MMEM:INIT:INT <NRf>
4) START FORMAT	MMEM:INIT "<msi>"
8) STORAGE CONFIG	————
1) INTERNAL RAM DISC	MMEM:MSI "RAM:"
2) INTERNAL DISC	MMEM:MSI "INT:"
3) EXTERNAL DISC	MMEM:MSI "EXT:"
7) FORMAT ASCII/BIN	MMEM:FORM ASC|BIN
8) PERIPHERL ADDRESSES	————
1) DISC ADDRESS	MMEM:MSI:ADDR <NRf>
2) DISC UNIT	MMEM:MSI:UNIT <NRf>
3) DISC VOLUME	MMEM:MSI:VOL <NRf>
4) PLOTTER ADDRESS	PLOT:ADDR <NRf>
5) PRINTER ADDRESS	PRIN:ADDR <NRf>
10) RETURN	————
9) CATALOG ON/OFF	SCR:CONT DCAT
10) RETURN	————
9) CATALOG ON/OFF	SCR:CONT DCAT
10) RETURN	————
User Define	————

Numeric Entry Group

FRONT PANEL KEY	HP-IB COMMAND
Marker Value	MARK[:A|:B|1|2]:VAL

Appendix C
HP-IB Command List

Common Commands (defined by IEEE 488.2)

*CAL?
*CLS
*ESE <NRf>
*ESE?
*ESR?
*IDN?
*OPC
*OPC?
*OPT?
*PCB <NRf>[,<NRf>]
*PSC <NRf>
*PSC?
*RST
*SRE <NRf>
*SRE?
*STB?
*TRG
*TST?
*WAI

Device-Specific Commands

ARM

ARM[:IMMediate]
ARM:SOURce {FREE|HOLD}
ARM:SOURce?

AVERage

AVER:COUNt <NRf>
AVER:COUNt?
AVER:DISPlay
    AVER:DISP:RATE <NRf>
    AVER:DISP:RATE?
    AVER:DISP:RATE
        AVER:DISP:RATE:STATe {OFF|ON|0|1}
        AVER:DISP:RATE:STATe?
AVER:INITialize
AVER:OVERlap <NRf> [PCT]
AVER:OVERlap?
AVER[:STATe] {OFF|ON|0|1}
AVER[:STATe]?
AVER:TYPE {PEAK|RMS|VECT}
AVER:TYPE?
AVER:WEIGhting {EXP|STAB}
AVER:WEIGhting?

CALibration

CAL[:ALL]?
CAL:AUTO {OFF|ON|0|1}
CAL:AUTO?
CAL:CLEar
CAL:SINGle
CAL:TRACe {OFF|ON|0|1}
CAL:TRACe?

CONFigure

CONF:TYPE {NETW|SPEC}
CONF:TYPE?

DISPlay[:A|:B|1|2]

DISP:DATA?
DISP:GRATicule {OFF|ON|0|1}
DISP:GRATicule?
DISP:HEADer
DISP:HEAD:AFORmat {ASC|FP32|FP64}
DISP:HEAD:AFORmat?
DISP:HEAD:NAME?
DISP:HEAD:POINts?
DISP:HEAD:PREamble?
DISP:HEAD:XINCrement?
DISP:HEAD:XNAMe?
DISP:HEAD:XORigin?
DISP:HEAD:XPOints?
DISP:HEAD:XUNits?
DISP:HEAD:YINCrement?
DISP:HEAD:YNAMe?
DISP:HEAD:YORigin?
DISP:HEAD:YPOints?
DISP:HEAD:YUNits?
DISP:LIMit
DISP:LIM:BEEPer {OFF|ON|0|1}
DISP:LIM:BEEPer?
DISP:LIM:FAIL
DISP:LIM:FAIL[:DATA]?
DISP:LIM:FAIL:HEADer
DISP:LIM:FAIL:HEAD:AFORmat {ASC|FP32|FP64}
DISP:LIM:FAIL:HEAD:AFORmat?
DISP:LIM:FAIL:HEAD:POINts?
DISP:LIM:LINE {OFF|ON|0|1}
DISP:LIM:LINE?
DISP:LIM:STATe {OFF|ON|0|1}
DISP:LIM:STATe?
DISP:LIM[:TABLe] <NRf>
DISP:LIM[:TABLe]?
DISP:LIM:TEST
DISP:LIM:TEST[:DATA]?
DISP:LIM:TEST:HEADer
DISP:LIM:TEST:HEAD:AFORmat {ASC|FP32|FP64}
DISP:LIM:TEST:HEAD:AFORmat?
DISP:LIM:TEST:HEAD:POINts?
DISP:X
DISP:X:APERture <NRf> [PCT]
DISP:X:APERture?
DISP:X:SPACing {LIN|LOG}
DISP:X:SPACing?

DISP[:Y]
DISP[:Y]:AXIS {GDEL|IMAG|MAGN|PHAS|REAL|LINM|LOGM}
DISP[:Y]:AXIS?
DISP[:Y]:SCALe
DISP[:Y]:SCAL:AUTO
DISP[:Y]:SCAL:AUTO:SINGle
DISP[:Y]:SCAL:CENTer <NRf>
DISP[:Y]:SCAL:CENTer?
DISP[:Y]:SCAL:DIVision <NRf>
DISP[:Y]:SCAL:DIVision?
DISP[:Y]:SCAL:REFerence {CENT|STAR|STOP|INP}
DISP[:Y]:SCAL:REFerence?
DISP[:Y]:SCAL:STARt <NRf>
DISP[:Y]:SCAL:STARt?
DISP[:Y]:SCAL:STOP <NRf>
DISP[:Y]:SCAL:STOP?
DISP[:Y]:SCAL:UNITs <string>
DISP[:Y]:SCAL:UNITs?
DISP[:Y]:SPACing {LIN|LOG}
DISP[:Y]:SPACing?

FREQuency

FREQ:CENTer <NRf>
FREQ:CENTer?
FREQ:CENTer
FREQ:CENT:STEP <NRf>
FREQ:CENT:STEP?
FREQ:REFerence {CENT|STAR}
FREQ:REFerence?
FREQ:SPAN <NRf>
FREQ:SPAN?
FREQ:SPAN
FREQ:SPAN:FULL
FREQ:STARt <NRf>
FREQ:STARt?

GPIB

GPIB:LEDS {OFF|ON|0|1}
GPIB:LEDS?
GPIB:MNEMonic {ECHO|OFF|SCR}
GPIB:MNEMonic?

INITialize

INIT:STATe {PAUS|RUN|STAR}
INIT:STATe?

INPut[1|2]

INP:COUPling {AC|DC}
INP:COUPling?
INP:IMPedance <NRf>
INP:IMPedance?
INP:LOW {FLO|GRO}
INP:LOW?
INP:RANGe <NRf>
INP:RANGe?
INP:RANGe
    INP:RANG:AUTO {OFF|ON|0|1}
    INP:RANG:AUTO?
INP:UNIT {EU|VOLT}
INP:UNIT?
INP:UNIT
    INP:UNIT:EU
        INP:UNIT:EU:MULTiplier <NRf>
        INP:UNIT:EU:MULTiplier?
        INP:UNIT:EU:NAME <string>
        INP:UNIT:EU:NAME?

LIMit[1-8]

LIMit:TABLe
    LIMit:TABL[:DATA] <block data>
    LIMit:TABL[:DATA]?
    LIMit:TABL:HEADer
        LIMit:TABL:HEAD:AFORmat {ASC|FP32|FP64}
        LIMit:TABL:HEAD:AFORmat?
        LIMit:TABL:HEAD:POINts?

MARKer[:A|:B|1|2]

MARK:BAND
MARK:BAND:CENTer <NRf>
MARK:BAND:CENTer?
MARK:BAND:POWer?
MARK:BAND:POWer
MARK:BAND:POW:STATe {OFF|ON|0|1}
MARK:BAND:POW:STATe?
MARK:BAND:STARt <NRf>
MARK:BAND:STARt?
MARK:BAND:STATe {OFF|ON|0|1}
MARK:BAND:STATe?
MARK:BAND:STOP <NRf>
MARK:BAND:STOP?
MARK:DTABle
MARK:DTAB[:DATA] <block data>
MARK:DTAB[:DATA]?
MARK:DTAB:HEADer
MARK:DTAB:HEAD:AFORmat {ASC|FP64|FP32}
MARK:DTAB:HEAD:AFORmat?
MARK:DTAB:HEAD:POINts?
MARK:DTAB:STATe {OFF|ON|0|1}
MARK:DTAB:STATe?
MARK:FUNC:AOFF
MARK:HARMonic
MARK:HARM:COUNt <NRf>
MARK:HARM:COUNt?
MARK:HARM[:FREQuency] <NRf>
MARK:HARM[:FREQuency]?
MARK:HARM:[FREQuency]
MARK:HARM:[FREQ]:DIVide <NRf>
MARK:HARM:POWer?
MARK:HARM:POWer
MARK:HARM:POW:STATe {OFF|ON|0|1}
MARK:HARM:POW:STATe?
MARK:HARM:STATe {OFF|ON|0|1}
MARK:HARM:STATe?
MARK:HARM:THD?
MARK:HARM:THD
MARK:HARM:THD:STATe {OFF|ON|0|1}
MARK:HARM:THD:STATe?
MARK:SIDeband
MARK:SID:COUNt <NRf>
MARK:SID:COUNt?
MARK:SID:DELTa <NRf>
MARK:SID:DELTa?
MARK:SID[:FREQuency] <NRf>
MARK:SID[:FREQuency]?
MARK:SID:POWer?

```
        MARK:SID:POWer
            MARK:SID:POW:STATe {OFF|ON|0|1}
            MARK:SID:POW:STATe?
        MARK:SID:STATe {OFF|ON|0|1}
        MARK:SID:STATe?
MARK[:X] <NRf>
MARK[:X]?
MARK[:X]
        MARK[:X]:AMAXimum
            MARK[:X]:AMAX:AUTO {OFF|ON|0|1}
            MARK[:X]:AMAX:AUTO?
            MARK[:X]:AMAX[:GLOBal]
            MARK[:X]:AMAX:LEFT
            MARK[:X]:AMAX:RIGHt
        MARK[:X]:AMINimum
            MARK[:X]:AMIN[:GLOBal]
        MARK[:X]:AMPLitude?
        MARK[:X]:AUTO {OFF|ON|0|1}
        MARK[:X]:AUTO?
        MARK[:X]:DELTa <NRf>
        MARK[:X]:DELTa?
        MARK[:X]:DELTa
            MARK[:X]:DELT:AMPLitude <NRf>
            MARK[:X]:DELT:AMPLitude?
            MARK[:X]:DELT:POINts <NRf>
            MARK[:X]:DELT:POIN?
            MARK[:X]:DELT:ZERO
        MARK[:X]:MODE {DELT|NORM}
        MARK[:X]:MODE?
        MARK[:X]:POINt <NRf>
        MARK[:X]:POINt?
        MARK[:X]:SEARch
            MARK[:X]:SEAR:AMPLitude <NRf>
            MARK[:X]:SEAR:AMPLitude?
            MARK[:X]:SEAR:LEFT
            MARK[:X]:SEAR:RIGHt
        MARK[:X]:STATe {OFF|ON|0|1}
        MARK[:X]:STATe?
```

MMEMory

MMEM:COPY <string>,<string>
MMEM:DELete <string>
MMEM:FORM {ASC|BIN}
MMEM:FORM?
MMEM:GET
    MMEM:GET:DTABle{:A|:B|1|2} <string>
    MMEM:GET:LIMit{limit#} <string>
    MMEM:GET:MATH <string>
    MMEM:GET:STATe <string>
    MMEM:GET:TRACe[:A|:B|1|2] <string>
MMEM:INITialize {"INT:"|"EXT:"|"RAM:"}
MMEM:INITialize
    MMEM:INIT:INTerleave <NRf>
    MMEM:INIT:INTerleave?
    MMEM:INIT:OPTion <NRf>
    MMEM:INIT:OPTion?
MMEM:LOAD
    MMEM:LOAD:APPLication <string>
        MMEM:LOAD:APPL:ALL {"INT!"|"EXT:"|"RAM"}
        MMEM:LOAD:APPL:AUTO {OFF|ON|0|1}
        MMEM:LOAD:APPL:AUTO?
    MMEM:LOAD:DTABle{:A|:B|1|2} <string>
    MMEM:LOAD:LIMit{limit#} <string>
    MMEM:LOAD:MATH <string>
    MMEM:LOAD:STATe <string>
    MMEM:LOAD:TRACe[:A|:B|1|2] <string>
MMEM:MSI
    MMEM:MSI:ADDRess <NRf>
    MMEM:MSI:ADDRess?
    MMEM:MSI:UNIT <NRf>
    MMEM:MSI:UNIT?
    MMEM:MSI:VOLume <NRf>
    MMEM:MSI:VOLume?
MMEM:MSI {"INT:"|"EXT:"|"RAM:"}
MMEM:MSI?
MMEM:REName <string>,<string>
MMEM:SAVE
    MMEM:SAVE:DTABle{:A|:B|1|2} <string>
    MMEM:SAVE:LIMit{limit#} <string>
    MMEM:SAVE:MATH <string>
    MMEM:SAVE:STATe <string>
    MMEM:SAVE:TRACe[:A|:B|1|2] <string>
MMEM:STORe
    MMEM:STOR:DTABle{:A|:B|1|2} <string>
    MMEM:STOR:LIMit{limit#} <string>
    MMEM:STOR:MATH <string>
    MMEM:STOR:STATe <string>
    MMEM:STOR:TRACe[:A|:B|1|2] <string>

PLOTter

PLOT:ADDRess <NRf>
PLOT:ADDRess?
PLOT:DUMP
    PLOT:DUMP:MARKer
    PLOT:DUMP:SCReen
    PLOT:DUMP:TRACe
PLOT:LTYPe
    PLOT:LTYP:TRAC[:A|:B|1|2] <NRf>
    PLOT:LTYP:TRAC[:A|:B|1|2]?
PLOT:PEN
    PLOT:PEN:ALPHa <NRf>
    PLOT:PEN:ALPHa?
    PLOT:PEN:GRID <NRf>
    PLOT:PEN:GRID?
    PLOT:PEN:INITialize
    PLOT:PEN:TRACe[:A|:B|1|2] <NRf>
    PLOT:PEN:TRACe[:A|:B|1|2]?
PLOT:SPEed <NRf>
PLOT:SPEed?

PRINter

PRIN:ADDRess <NRf>
PRIN:ADDRess?
PRIN:DUMP
    PRIN:DUMP:ALPHa
    PRIN:DUMP:SCReen

SCReen

SCR:ACTive {A|B}
SCR:ACTive?
SCR:ANNotation {OFF|ON|0|1}
SCR:ANNotation?
SCR:CONTents {APPL|DCAT|FLOG|MEM|STAT|TLOG|TRAC}
SCR:CONTents?
SCR:FORMat {FBAC|SING|ULOW}
SCR:FORMat?
SCR[:STATe] {OFF|ON|0|1}
SCR[:STATe]?

SERVice

SERV:ADC2
    SERV:ADC2:PAS2 {OFF|ON|1|0}
SERV:ADJ
    SERV:ADJ:ADC
        SERV:ADJ:ADC:GAIN
        SERV:ADJ:ADC:OFFS
    SERV:ADJ:CMRR[1|2]
    SERV:ADJ:FLAT1
        SERV:ADJ:FLAT1:CENT
        SERV:ADJ:FLAT1:FULL
    SERV:ADJ:FLAT2
    SERV:ADJ:OFFS[1|2]
SERV:CAL
    SERV:CAL:HIGH {OFF|ON|0|1}
    SERV:CAL:HIGH?
    SERV:CAL:LOW {OFF|ON|0|1}
    SERV:CAL:LOW?
SERV:TRACk {OFF|ON|0|1}
SERV:TRACk?
SERV:DITH {OFF|ON|0|1}
SERV:DITH?
SERV:INP[1|2]
    SERV:INP[1|2]:OFFS <NRf>
SERV:SHOR[1|2] {OFF|ON|1|0}
SERV:SOUR
    SERV:SOUR:INP {OFF|ON|0|1}
    SERV:SOUR:INP?

SOURce

SOUR:AMPLitude
    SOUR:AMPL[:LEVel] <NRf>
    SOUR:AMPL[:LEVel]?
SOUR:FREQuency
    SOUR:FREQ[:CW] <NRf>
    SOUR:FREQ[:CW]?
    SOUR:FREQ:MODE {RAND|CW|PCH}
    SOUR:FREQ:MODE?
SOUR:STATe {OFF|ON|0|1}
SOUR:STATe?

STATus

STAT:DEVice
- STAT:DEV:CONDition?
- STAT:DEV:ENABle <NRf>
- STAT:DEV:ENABle?
- STAT:DEV:EVENt?
- STAT:DEV:NTR <NRf>
- STAT:DEV:NTR?
- STAT:DEV:PTR <NRf>
- STAT:DEV:PTR?

STAT:DINTegrity
- STAT:DINT:CONDition?
- STAT:DINT:ENABle <NRf>
- STAT:DINT:ENABle?
- STAT:DINT:EVENt?
- STAT:DINT:NTR <NRf>
- STAT:DINT:NTR?
- STAT:DINT:PTR <NRf>
- STAT:DINT:PTR?

STAT:USER
- STAT:USER:ENAB le<NRf>
- STAT:USER:ENABle?
- STAT:USER:EVENt?
- STAT:USER:PULSe <NRf>

SWEep

SWE:TIME <NRf>
SWE:TIME?

SYSTem

SYST:ADDRess <NRf>
SYST:ADDRess?
SYST:BEEPer {OFF|ON|0|1}
SYST:BEEPer?
SYST:DATE <NRf>,<NRf>,<NRf>
SYST:DATE?
SYST:DTIMe <NRf>,<NRf>,<NRf>,<NRf>,<NRf>,<NRf>
SYST:DTIMe?
SYST:DUMP
    SYST:DUMP:PLOTer <NRf>
    SYST:DUMP:PLOTer?
    SYST:DUMP:PRINter <NRf>
    SYST:DUMP:PRINter?
SYST:ERRor?
SYST:FLOG?
SYST:FLOG
    SYST:FLOG:CLEar
    SYST:FLOG:ENTRY
        SYST:FLOG:ENTR:DESCription <NRf>
SYST:HEADer {OFF|ON|0|1}
SYST:HEADer?
SYST:MEMory?
SYST:SAVE
SYST:SERial?
SYST:SET <block data>
SYST:SET?
SYST:SET
    SYST:SET:FORM {ASC|BIN}
    SYST:SET:FORM?
SYST:TIME <NRf>,<NRf>,<NRf>
SYST:TIME?

TEST

- TEST:ABOR
- TEST:ALL
- TEST:DISP
  - TEST:DISP:PATT {OFF|ON|0|1}
  - TEST:DISP:PATT?
- TEST:DMA
- TEST:DSP
  - TEST:DSP:ALL
  - TEST:DSP:BASE
  - TEST:DSP:FIFO
  - TEST:DSP:FILT
  - TEST:DSP:LO
  - TEST:DSP:SOUR
  - TEST:DSP:TRIG
  - TEST:DSP:ZOOM
- TEST:IO
  - TEST:IO:ALL
  - TEST:IO:DISC
    - TEST:IO:DISC:ALL
    - TEST:IO:DISC:CONT
    - TEST:IO:DISC:MOT
    - TEST:IO:DISC:RAND
    - TEST:IO:DISC:READ
    - TEST:IO:DISC:REST
    - TEST:IO:DISC:RW
    - TEST:IO:DISC:SEEK <NRf>
    - TEST:IO:DISC:WRIT
  - TEST:IO:FBUS
  - TEST:IO:KEY
  - TEST:IO:SBUS
- TEST:KEY?
- TEST:LCONfidence
  - TEST:LCON:RESult?
- TEST:LOG?
- TEST:LOG
  - TEST:LOG:CLE
- TEST:LOOP
  - TEST:LOOP:STARt
  - TEST:LOOP[:STAT] {OFF|ON|0|1}
  - TEST:LOOP[:STAT]?
- TEST:MATH
- TEST:PROC
  - TEST:PROC:ALL
  - TEST:PROC:CPU
  - TEST:PROC:INT
  - TEST:PROC:MFP
  - TEST:PROC:RAM
  - TEST:PROC:ROM
- TEST:REC
  - TEST:REC:GARR

TEST:REC:INP
TEST:RESult?
TEST:SHORt
TEST:SOURce
TEST:SOUR:ALL
TEST:SOUR:BASE
TEST:SOUR:CPU
TEST:SOUR:LO
TEST:SOUR:ZOOM

TRACe[:A|:B|1|2]

TRAC:DATA <block data>
TRAC:DATA?
TRAC:DATA
TRAC:DATA:SET <block data>
TRAC:DATA:SET?
TRAC:HEADer
TRAC:HEAD:AFORmat {ASC|FP32|FP64}
TRAC:HEAD:AFORmat?
TRAC:HEAD:NAME <string>
TRAC:HEAD:NAME?
TRAC:HEAD:POINts <NRf>
TRAC:HEAD:POINts?
TRAC:HEAD:PREamble?
TRAC:HEAD:XINCrement <NRf>
TRAC:HEAD:XINCrement?
TRAC:HEAD:XNAMe?
TRAC:HEAD:XORigin <NRf>
TRAC:HEAD:XORigin?
TRAC:HEAD:XPOints <NRf>
TRAC:HEAD:XPOints?
TRAC:HEAD:XUNits?
TRAC:HEAD:YINCrement <NRf>
TRAC:HEAD:YINCrement?
TRAC:HEAD:YNAMe?
TRAC:HEAD:YORigin <NRf>
TRAC:HEAD:YORigin?
TRAC:HEAD:YPOints <NRf>
TRAC:HEAD:YPOints?
TRAC:HEAD:YUNits?
TRAC:RESult {COH|CSP|F{1-5}|FRES|K{1-5}|PSD{1|2}|SPEC{1|2}|TIME{1|2}}
TRAC:RESult?
TRAC:TITLe <string>
TRAC:TITLe?

TRIGger

TRIG:DELay[1|2] <NRf>
TRIG:DELay[1|2]?
TRIG[:IMMediate]
TRIG:LEVel <NRf> [PCT]
TRIG:LEVel?
TRIG:SLOPe {NEG|POS}
TRIG:SLOPe?
TRIG:SOURce {BUS|FREE|EXT|INT{1|2}|SOUR}
TRIG:SOURce?

USER

USER:EXPRession {F1|F2|F3|F4|F5},<expr>
USER:EXPRession? {F1|F2|F3|F4|F5}
USER:VARiable
    USER:VAR:IMAG {K1|K2|K3|K4|K5},<NRf>
    USER:VAR:IMAG? {K1|K2|K3|K4|K5}
    USER:VAR:MAGNitude {K1|K2|K3|K4|K5},<NRf>
    USER:VAR:MAGNitude? {K1|K2|K3|K4|K5}
    USER:VAR:PHASe {K1|K2|K3|K4|K5},<NRf> [RAD|DEG]
    USER:VAR:PHASe? {K1|K2|K3|K4|K5}
    USER:VAR:REAL {K1|K2|K3|K4|K5},<NRf>
    USER:VAR:REAL? {K1|K2|K3|K4|K5}

WINDow[1|2]

WIND:CONStant
    WIND:CONS:EXPonential <NRf>
    WIND:CONS:EXPonential?
    WIND:CONS:FORCe <NRf>
    WIND:CONS:FORCe?
WIND[:TYPE] {FLAT|HANN|UNIF|FORC|EXP}
WIND[:TYPE]?

Appendix D
Error Messages

Introduction

This appendix lists error messages generated by the HP 35660A. As many as ten messages can be stored in the analyzer's error queue. One message is read from the queue each time you send the SYST:ERR query. Errors are returned in the order generated.

NOTE When queried with SYST:ERR?, the HP 35660A returns a maximum of 255 characters. It truncates any error messages greater than 255 characters before putting them on the HP-IB.

Command Errors

-100 CMD ERR

Disc controller error
Empty function redefinition not allowed
Function definition too long, split into two functions
Incompatible mix of operand types
Lower function number required
Not a valid message unit delimiter
Parameter cannot be decremented
Parameter cannot be incremented
Unbalanced left and right parentheses
YOU PRESSED <num> KEYS IN <string>
<num> execution exceeds memory limitations

-101 INVALID CHAR

-110 BAD CMD

Command is not a query: '<string>'
Unknown command: '<string>'

-120 BAD PARM

Block data contains bad parameters
Block data expected
Data block too long (truncating)
Definite block data expected
Eng Label syntax error, unbalanced () chars, previous label retained
Eng Label syntax error, ^ not followed by number or more than one ^, previous label retained
Header format not recognized
Illegal decimal point
Illegal exponent
Index too large, moving to largest
Index too small, moving to smallest
Not a valid choice
Not a valid data block: '<string>'
Not a valid expression: '<string>'
Not a valid number and/or units
Not a valid number
Not a valid number: '<string>'
Not a valid parameter : <string>
Not a valid stored data filename : <string>
Not a valid string: '<string>'
Not a valid suffix '<string>'
Not a valid unit
Too many characters.
Too many digits
Wrong type of parameter: '<string>'
<string>

-123 OVERFLOW

-129 PARM MISSING

Missing parameter

-142 TOO MANY PARMS

Command is query only
Incorrect number of parameters
No entry allowed
Parameter is inactive

Execution Errors

-200 EXECUTE ERROR

Application load aborted, '<string>' is already loaded
Application load aborted, '<string>' is auto-load only
Attempt to go beyond end of storage
Cannot change units
Cannot format a single sided disc
Copy aborted, Destination does not have space
Copy aborted Source and Destination media are the same
COPY DISC with same source and destination can not be executed over HP-IB
Disc format aborted
Disc operation aborted
Disc operation already in progress
Disc operation failed
File '<string>' does not contain <string>
Full directory
Not a valid base node
Not a valid directory name
Not a valid directory
Not a valid file name
Not a valid file type
Not a valid volume, press CONTINUE COPY when ready, press ABORT COPY to quit
Not ready, press CONTINUE COPY when ready, press ABORT COPY to quit
Please PAUSE triggered measurement to use Internal Disc
Plot / Print Aborted
Plot already in progress, press ABORT to stop
Print already in progress, press ABORT to stop
Re-size failure
Requested size exceeds <num> bytes available
Serial number has not been defined
Size too big, maximum is <num> bytes
Storage device busy
Target not found
Terminated due to device error
This label reserved for display, previous label retained
Unable to delete due to open file
Unable to read file '<string>'
Unable to read file '<string>'. Instrument has been preset
Unable to write file '<string>'
<num> definition is not valid for execution

-203 TRIGGER ERROR

-211 SETTINGS CONFLICT

ARM is automatic, key ignored
Band markers allowed only in frequency domain
CHANNEL 2 TRIGGER not permitted in MEAS TYPE 1 CHANNEL.
COHERENCE data is not valid with PEAK HOLD AVERAGE
COHERENCE data is not valid with VECTOR AVERAGE
CROSS SPECTRUM data is not valid with PEAK HOLD AVERAGE
CROSS SPECTRUM data is not valid with PEAK HOLD AVERAGE
FREQUENCY RESPONSE data is not valid with PEAK HOLD AVERAGE
FREQUENCY RESPONSE data is not valid with PEAK HOLD AVERAGE
Harmonic markers allowed only in frequency domain
Log X axis not allowed with X axis values less than 0
Main marker is off
Marker results not available with this measurment data
Marker results not available with this trace type
Marker value not a valid entry
No limit table is associated with this trace Use SELECT LIMIT key
No valid results are available
Reference level must be less than 0 dB in Coherence
Reference level must be less than 1 in Coherence
Reference level tracking not allowed on this data
SCR:CONT FLOG must be sent first
Select AVERAGE ON before COHERENCE data
Select AVERAGE ON to view COHERENCE data
Select AVERAGE TYPE RMS to view COHERENCE data
Select AVERAGE TYPE RMS to view COHERENCE data
Select MEAS TYPE '2 Channel' to view channel 2 PSD data
Select MEAS TYPE '2 Channel' to view channel 2 spectrum data
Select MEAS TYPE '2 Channel' to view channel 2 TIME data
Select MEAS TYPE '2 Channel' to view COHERENCE data
Select MEAS TYPE '2 Channel' to view CROSS SPECTRUM data
Select MEAS TYPE '2 Channel' to view FREQUENCY RESPONSE data
Serial number already set
Sideband markers allowed only in frequency domain
This trace type is not allowed with the current MEAS DATA selection
<num> execution requires unavailable measurement data operand : <string> Change instrument setup to correct
<num> execution requires unavailable stored data operand : <string> Change instrument setup to correct

-212 OUT OF RANGE

Address must be between 0 and 30
Bottom of scale must be positive
Bottom of scale out of range, value not accepted
Center of scale must be positive
Center of scale out of range, value not accepted
dBm Ref Impedance cannot be <= 0
Directory name limited to <num> characters
Disc does not support specified format option
Disc does not support specified interleave
Eng Label cannot be a space string, previous label retained
Eng Label too long, shortened to 8 characters
Eng Unit Value cannot be = 0, previous value retained
Exceeded maximum number of segments
File name limited to <num> characters
Interleave too large
Line type must be between 0 and 6
Marker X entry must be within span
No fault log entry at position <num>
Non-printing characters are not allowed
Not a valid date, system date not reset
Not a valid entry, nearest acceptable value selected
Not a valid format option for the internal disc
Not a valid security code
Not a valid serial number
Not a valid time, system time not reset
Null file names are not allowed
Out of range
Out of range
Pen number must be between 0 and 64
Perdiv out of range, value not accepted
Plot speed must be between 1 and 100
Selected limit table out of range
Serial number must be 10 characters
Top of scale must be positive
Top of scale out of range, value not accepted
Valid disc unit values are 0 to 15
Valid disc volume values are 0 to 7
Valid HP-IB addresses are 0 to 30
Valid HP-IB disc addresses are 0 to 7

-222 OUT OF MEMORY

Application load of '<string>' aborted, Not enough memory
Insufficient memory to add new segment
Insufficient memory to recall data table
Insufficient memory to recall limit table
Not enough memory for <string>
Unable to allocate copy buffer

-240 MASS STORAGE ERROR

Application load of '<string>' aborted, Mass storage failure
Auto sparing invoked
Bad disc request status
Bad disc status
Bad disc
Block address too large
Cannot execute until diagnostic release
Cannot execute until internal maintenance release
Cannot execute until operator release
Channel parity error
Cross-unit error during copy data
Diagnostic failed
Diagnostic release requested
Disc error
Disc removed when files open, files are now closed
Disc Timeout
Hardware fault in controller
Hardware fault in unit
Illegal opcode
Illegal parallel operation
Internal maintenance release requested
Latency induced
Message length wrong
Message sequence violated
Operator release requested
Parameter out of range
Parameter wrong length
Retry transaction

-241 HARDWARE MISSING

Disc controller error
Disc drive error
Illegal volume or unit number
Plot / Print device not present
Unrecognized disc drive

-242 NO MEDIA

Disc not in drive
Not ready (No media)
Storage device not found

-243 BAD MEDIA

Application load aborted, '<string>' is not an application
Bad media
Disc CRC error
End of file
End of volume
Maintenance track overflow
Marginal data
Media wear (1 spare left)
More than one unrecoverable data block
No data in a block
No spare tracks left
Recoverable data overflow
Recoverable data
Unformatted media
Unrecoverable data block

-244 MEDIA FULL

Insufficient disc space

-245 DIR FULL

-246 FILE NAME NOT FOUND

Application load aborted, File '<string>' not found
File not found
File '<string>' not found
No new applications found

-247 DUPLICATE NAME

Duplicate file name

-248 MEDIA PROTECTED

Application load aborted, File '<string>' is protected
Write protect
Write protected disc

Internal Errors

-300 INTERNAL ERROR

Auto-zero fails Chan:<num> Range: <num> dBVrms
DMA Timeout
EEPROM clear failed to zero all bytes
EEPROM not initialized correctly
Front panel key (#<num>) is stuck
IIC (slow bus) failed on power up
Keyboard test ..... fails
Memory option failed. Instrument has been re-configured to use <num> megabytes
No data received during cal
No source trigger received during cal
Power on tests fail, for details please see: <Special Fctn> <Self Test> <Test Log>
Save system configuration to EEPROM failed

-302 SYSTEM ERROR

See SPECIAL FCTN Fault Log

-303 TIME OUT ERROR

Control not requested (control passed back)
Controller did not pass control
Controller did not receive control back

-310 MEMORY ERROR

-313 CAL DATA LOSS

Calibration fails: <string>

-330 SELF TEST ERROR

-350 TOO MANY ERRORS

Query Errors

-400 QUERY ERROR

-410 INTERRUPTED

Query interrupted

-420 UNTERMINATED

Command unterminated near '<string>'; addressed to talk with nothing to say

-422 ADDR TALK NO OUTPUT

-430 DEADLOCK

HP-IB deadlocked, output buffer cleared

Index

D

G

H

I

L

M

N

O

P

Q

R

S

T

U

V

W

X

Y

Z

Hewlett-Packard Sales and Service Offices

To obtain Servicing information or to order replacement parts, contact the nearest Hewlett-Packard Sales and Service Office listed in HP Catalog, or contact the nearest regional office listed below:

In the United States

California
P.O. Box 4230
1421 South Manhattan Avenue
Fullerton 92631

Georgia
P.O. Box 105005
2000 South Park Place
Atlanta 30339

Illinois
5201 Tollview Drive
Rolling Meadows

New Jersey
W. 120 Century Road
Paramus 07652

In Canada
Hewlett-Packard (Canada) Ltd.
17500 South Service Road
Trans-Canada Highway
Kirkland, Quebec H9J 2M5

In France
Hewlett-Packard France
F-91947 Les Ulis Cedex
Orsay

In German Federal Republic
Hewlett-Packard GmbH
Vertriebszentrale Frankfurt
Berner Strasse 117
Postfach 560 140
D-6000 Frankfurt 56

In Great Britain
Hewlett-Packard Ltd.
King Street Lane
Winnersh, Wokingham
Berkshire RG11 5AR

In Other European Countries
Switzerland
Hewlett-Packard (Schweiz) AG
7, rue du Bois-du-Lan
Case Postale 365
CH-1217 Meyrin

In All Other Locations
Hewlett-Packard Inter-Americas
3155 Porter Drive
Palo Alto, California 94304