Programmer's Guide

HP 70703A Digitizing Oscilloscope

HEWLETT PACKARD

HP Part No. 70703-90029
Printed in United Kingdom December 1993

Contents

Figures

Tables

Introduction to Programming Syntax

This chapter introduces you to the basic concepts of HP-IB communication and provides information and examples to get you started programming. The exact mnemonics for the commands are listed in Chapters 5 through 18.

Talking to the Instrument

In general, computers acting as controllers communicate with the instrument by passing messages over a remote interface using the I/O statements provided in the instruction set of the controller's host language. Hence, the messages for programming the HP 70703A, described in this manual, will normally appear as ASCII character strings imbedded inside the I/O statements of your controller's program. For example, the HP 9000 Series 200/300 BASIC and PASCAL language systems use the OUTPUT statement for sending program messages to the HP 70703A, and the ENTER statement for receiving response messages from the HP 70703A. Most commands and parameters are compatible with the HP 54503A and HP E1426A osilloscopes.

Messages are placed on the bus using an output command and passing the device address, program message, and terminator. Passing the device address ensures that the program message is sent to the correct interface and instrument.

The following command presets the HP 70703A:

```
OUTPUT <device address>;"*RST"<terminator>
```

<device address> represents the address of the device being programmed.

Note

The programming examples in this manual are written in HP BASIC 5.0 for an HP 9000 Series 200/300 controller.

The actual OUTPUT command you use when programming is dependent on the controller and the programming language you are using.

Angular brackets "< >" in this manual, enclose words or characters that symbolize a program code parameter or a bus command. Information that is displayed in quotes represents the actual message that is sent across the bus. The message terminator (NL or EOI) is the only additional information that is also sent across the bus.

For HP 9000 Series 200/300 controllers, it is not necessary to type in the actual <terminator> at the end of the program message. These controllers automatically terminate the program message internally when the return key is pressed.

Addressing the Instrument

Since HP-IB can address multiple devices through the same interface card, the device address passed with the program message must include not only the correct interface select code, but also the correct instrument address.

Interface Select Code (Selects Interface). Each interface card has a unique interface select code. This code is used by the controller to direct commands and communications to the proper interface. The default is typically “7” for HP-IB controllers.

Instrument Address (Selects Instrument). Each instrument on an HP-IB bus must have a unique instrument address between decimal 0 and 30. The device address passed with the program message must include not only the correct instrument address, but also the correct interface select code.

```
   DEVICE ADDRESS = (Interface Select Code * 100) + (Instrument Address)
```

For example, if the instrument address for the HP 70703A is 4 and the interface select code is 7, when the program message is passed, the routine performs its function on the instrument at device address 704.

For the HP 70703A, the instrument address is typically set to “7” at the factory. Consult the “Installation and Verification Manual” for details on how to change the instrument address.

Note The program examples in this manual assume the HP 70703A is at device address 707.

Program Message Syntax

To program the HP 70703A over the bus, you must have an understanding of the command format and structure expected by the instrument. The instrument is remotely programmed with program messages. These are composed of sequences of program message units, with each unit representing a program command or query. A program command or query is composed of a sequence of functional elements that includes separators, headers, program data, and terminators. These are sent to the instrument over the system interface as a sequence of ASCII data messages. For example:

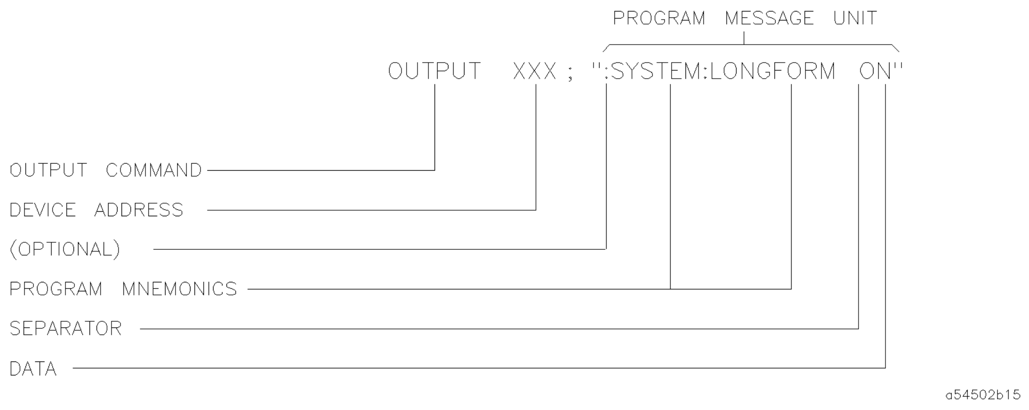


Figure 1-1.

Separator

The <separator> shown in the program message refers to a blank space which is required to separate the program mnemonic from the program data.

Command Syntax

A command is composed of a header, any associated data, and a terminator. The header is the mnemonic or mnemonics that represent the operation to be performed by the instrument. The different types of headers are discussed in the following paragraphs.

Simple Command Header

Simple command headers contain a single mnemonic. AUTOSCALE and DIGITIZE are examples of simple command headers typically used in this instrument. The syntax is:

```
<program mnemonic><terminator>
```

When program data must be included with the simple command header (for example, :DIGITIZE CHAN1), a separator is added. The syntax is:

```
<program mnemonic><separator><program data>
<terminator>
```

Compound Command Header

Compound command headers are a combination of two or more program mnemonics. The first mnemonic selects the subsystem, and the last mnemonic selects the function within that subsystem. Additional mnemonics appear between the subsystem mnemonic and the function mnemonic when there are additional levels within the subsystem that must be transversed. The mnemonics within the compound message are separated by colons. For example:

To execute a single function within a subsystem, use the following:

```
:<subsystem>:<function><separator>
<program data><terminator>
```

(For example, :SYSTEM:LONGFORM ON)

To transverse down a level of a subsystem to execute a subsystem within that subsystem:

```
:<subsystem>:<subsystem>:<function>
<separator><program data><terminator>
```

(For example, :TRIGGER:DELAY:SOURCE CHAN1)

To execute more than one function within the same subsystem, a a semicolon is used to separate the functions:

```
:<subsystem>:<function><separator>
<data>;<function><separator><data>
<terminator>
```

(For example, :SYSTEM:LONGFORM ON;HEADER OFF)

Identical function mnemonics can be used for more than one subsystem. For example, the function mnemonic RANGE may be used to change the vertical range or to change the horizontal range:

```
:CHANNEL1:RANGE .4
```

- sets the vertical range of channel 1 to 0.4 volts full scale.

```
:TIMEBASE:RANGE 1
```

- sets the horizontal timebase to 1 second full scale. CHANNEL1 and TIMEBASE are subsystem selectors and determine which range is being modified.

Common Command Header

Common command headers control IEEE 488.2 functions within the instrument (such as clear status, and so on). Their syntax is:

```
*<command header><terminator>
```

No space or separator is allowed between the asterisk and the command header. *CLS is an example of a common command header.

Query Command

Command headers immediately followed by a question mark (?) are queries. After receiving a query, the instrument interrogates the requested function and places the answer in its output queue. The answer remains in the output queue until it is read or another command is issued. When read, the answer is transmitted across the bus to the designated listener (typically a controller).

The query :TIMEBASE:RANGE? places the current timebase setting in the output queue. The controller input statement:

```
ENTER <device address>;Range
```

passes the value across the bus to the controller and places it in the variable Range.

Query commands are used to find out how the instrument is currently configured. They are also used to get results of measurements made by the instrument, with the query actually activating the measurement. For example, the query :MEASURE:RISETIME? instructs the instrument to measure the risetime of your waveform and place the result in the output queue.

Note The output queue must be read before the next program message is sent. For example, when you send the query :MEASURE:RISETIME? you must follow that query with the program statement ENTER Value_risetime to read the result of the query and place the result in a variable (Value_risetime). Sending another command before reading the result of the query will cause the output buffer to be cleared and the current response to be lost. This will also generate an error in the error queue.

Program Header Options

Program headers can be sent using any combination of uppercase or lowercase ASCII characters. Instrument responses, however, are always returned in uppercase.

Both program command and query headers may be sent in either longform (complete spelling), shortform (abbreviated spelling), or any combination of longform and shortform. Either of the following examples sets the vertical range for:

```
:CHANNEL1:RANGE 1.2 (longform)

:CHAN1:RANG 1.2  (shortform)
```

Programs written in longform are easily read and are almost self-documenting. The shortform syntax conserves the amount of controller memory needed for program storage and reduces the amount of I/O activity.

Note The rules for shortform syntax are shown in the chapter "Programming and Documentation Conventions."

Program Data

Program data is used to convey a variety of types of parameter information related to the command header. At least one space must separate the command header or query header from the program data.

```
<program mnemonic><separator><data>
<terminator>
```

When a program mnemonic or query has multiple data parameters a comma separates sequential program data.

```
<program mnemonic><separator><data>,
<data><terminator>
```

For example, :TRIGGER:DELAY TIME,1.23E-01 has two data parameters: TIME and 1.23E-01.

Character Program Data

Character program data is used to convey parameter information as alpha or alphanumeric strings. For example, the timebase command MODE can be set to auto, trigger, or single. The character program data in this case may be AUTO, TRIGGER, or SINGLE. :TIMEBASE:MODE SINGLE sets the timebase mode to single.

Numeric Program Data

Some command headers require program data to be a number. For example, :TIMEBASE:RANGE requires the desired full scale range to be expressed numerically. The instrument recognizes integers, real numbers, and scientific notation. For more information see the appendix "Message Communication and System Functions."

Program Message Terminator

The program codes within a data message are executed after the program message terminator is received.

The terminator may be either an NL (New Line) character, an EOI (End-Or-Identify) asserted, or a combination of the two. All three ways are equivalent with the exact encodings for the program terminators listed in the appendix "Message Communication and System Functions." Asserting the EOI sets the EOI control line low on the last byte of the data message. The NL character is an ASCII linefeed (decimal 10).

Note The NL (New Line) terminator has the same function as an EOS (End-Of-String) and EOT (End-Of-Text) terminator.

Selecting Multiple Subsystems

You can send multiple program commands and program queries for different subsystems on the same line by separating each command with a semicolon. The colon following the semicolon enables you to enter a new subsystem. For example:

```
<program mnemonic><data>;:<program mnemonic>
<data><terminator>

:CHANNEL1:RANGE 0.4;:TIMEBASE RANGE 1
```

Note Multiple commands may be any combination of compound and simple commands.

Summary

The following illustration summarizes the syntax for programming over the bus.

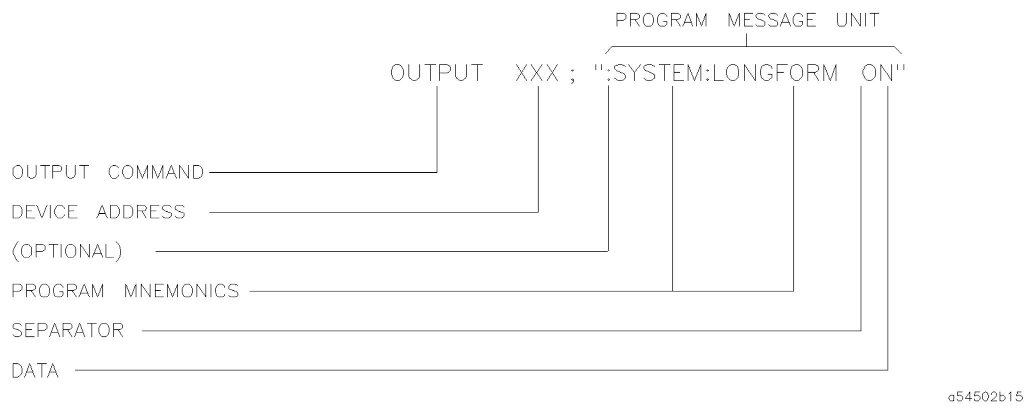


Figure 1-2.

2

Introduction to Programming an Instrument

There are four basic operations that can be done with a controller and an oscilloscope via HP-IB. You can:

1. Set up the instrument and start measurements.
2. Retrieve setup information and measurement results.
3. Digitize a waveform and pass the data to the controller.
4. Send measurement data to the instrument.

Other more complicated tasks are accomplished with a combination of these four basic functions.

This chapter deals mainly with how to set up the instrument, how to retrieve setup information and measurement results, how to digitize a waveform, and how to pass data to the controller. Refer to the chapter "Measure Subsystem" for information on sending measurement data to the instrument.

Note The programming examples in this manual are written in HP BASIC 5.0 for an HP 9000 Series 200/300 controller.

Initialization

To make sure the bus and all appropriate interfaces are in a known state, begin every program with an initialization statement. For example:

```
CLEAR 707  ! initializes the interface of the instrument.
```

Then initialize the instrument to a preset state. For example:

```
OUTPUT 707;"*RST"  ! initializes the instrument to a preset state.
```

Note The actual commands and syntax for initializing the instrument are discussed in the chapter "Common Commands."

Refer to your controller manual and programming language reference manual for information on initializing the interface.

Autoscale

The AUTOSCALE feature of Hewlett-Packard digitizing oscilloscopes performs a very useful function on unknown waveforms by setting up the vertical channel, timebase, and trigger level of the instrument. The syntax for AUTOSCALE is:

```
:AUTOSCALE<terminator>
```

Setting Up the Instrument

A typical oscilloscope setup would set the vertical range and offset voltage, the horizontal range, delay time, delay reference, trigger mode, trigger level, and slope. A typical example of the commands sent to the oscilloscope are:

```
:CHANNEL1:RANGE 0.64;OFFSET 0.25<terminator>
:TIMEBASE:RANGE 1E-3;DELAY 20E-9;MODE TRIGGERED<terminator>
:TRIGGER:LEVEL 0.25;SLOPE POSITIVE<terminator>
```

This example sets the vertical to 0.64 V full-scale (80 mV/div) centered at 0.25 V. The horizontal time is 1 ms full-scale with 20 ns delay. The timebase mode is set to triggered, and the trigger circuit is programmed to trigger at 0.25 V on a positive slope.

Receiving Information from the Instrument

After receiving a query (command header followed by a question mark), the instrument interrogates the requested function and places the answer in its output queue. The answer remains in the output queue until it is read or another command is issued. When read, the answer is transmitted across the bus to the designated listener (typically a controller). The input statement for receiving a response message from an instrument's output queue typically has two parameters; the device address and a format specification for handling the response message. For example, to read the result of the query command :SYSTEM:LONGFORM?, you would execute the statement:

```
ENTER <device address>;Setting$
```

where <device address> represents the address of your device. This would enter the current setting for the longform command in the string variable Setting$.

Note All results for queries sent in a program message must be read before another program message is sent. For example, when you send the query :MEASURE:RISETIME?, you must follow that query with the program statement ENTER Risetime$ to read the result of the query and place the result in a variable (Risetime$).

Sending another command before reading the result of the query will cause the output buffer to be cleared and the current response to be lost. This will also cause an error to be placed in the error queue.

Executing an ENTER statement before sending a query will cause the controller to wait indefinitely.

Note The actual ENTER program statement you use when programming is dependent on the programming language you are using.

The format specification for handling the response messages is dependent on both the controller and the programming language.

Response Header Options

The format of the returned ASCII string depends on the current setting of the SYSTEM LONGFORM command.

```
<data><terminator>
```

For example, with **:SYSTEM:LONGFORM ON**, **:CHANnel1:COUPling?** returns DCFIFTY, and with **:SYSTEM:LONGFORM OFF**, DCF is returned.

Note A command or query may be sent in either longform or shortform, or in any combination of longform and shortform. The LONGFORM command only controls the format of the returned data and has no effect on the way commands are sent. Common commands never return a header.

Refer to the chapter "System Subsystem" for information on turning the LONGFORM command on and off.

Response Data Formats

Most data will be returned as exponential or integer numbers. However, query data of instrument setups may be returned as character data. Interrogating the trigger SLOPE? will return one of the following:

```
POSITIVE<terminator> (with LONGFORM ON)

POS<terminator>      (with LONGFORM OFF)
```

Note Refer to the individual commands in this manual for information on the format (alpha or numeric) of the data returned from each query.

String Variables

Reading queries into string variables is simple and straightforward, requiring little attention to formatting. For example:

```
ENTER <device address>;Result$
```

places the output of the query in the string variable Result$.

Note String variables are case sensitive and must be expressed exactly the same each time they are used.

Note The output of the instrument may be numeric or character data depending on what is queried. Refer to the specific commands for the formats and types of data returned from queries.

Note For the example programs, assume that the device being programmed is at device address 707. The actual address will vary according to how you have configured the bus for your own application.

The following example shows the data being returned to a string variable:

```
10 DIM Rang$[30]
20 OUTPUT 707;":CHANNEL1:RANGE?"
30 ENTER 707;Rang$
40 PRINT Rang$
50 END
```

After running this program, the controller displays: +1.00000E-1

Numeric Variables

Numeric variables can be used when the query data is all numeric.

The following example shows the data being returned to a numeric variable.

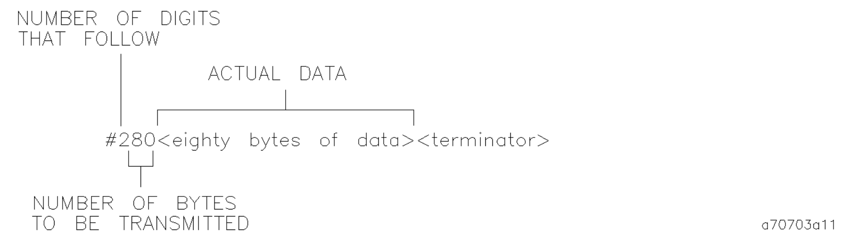
```
10 OUTPUT 707;":SYSTEM:HEADER OFF"
20 OUTPUT 707;":CHANNEL1:RANGE?"
30 ENTER 707;Rang
40 PRINT Rang
50 END
```

After running this program, the controller displays: .1

Definite-Length Block Response Data

Definite-length block response data allows any type of device-dependent data to be transmitted over the system interface as a series of 8-bit binary data bytes. This is particularly useful for sending large quantities of data or 8-bit extended ASCII codes. The syntax is a hash sign (#) followed by a non-zero digit representing the number of digits in the decimal integer. After the non-zero digit is the decimal integer that states the number of 8-bit data bytes being sent. This is followed by the actual data.

For example, for transmitting 80 bytes of data, the syntax would be:



The “2” states the number of digits that follow, and “80” states the number of bytes to be transmitted.

Multiple Queries

You can send multiple queries to the instrument within a single program message, but you must also read them back within a single program message. This can be accomplished by either reading them back into a string variable or into multiple numeric variables. For example, you could read the result of the query :TIMEBASE:RANGE?;DELAY? into the string variable Results$ with the command:

```
ENTER 707;Results$
```

When you read the result of multiple queries into string variables, each response is separated by a semicolon. For example, the response of the query :TIMEBASE:RANGE?;DELAY? would be:

```
<range_value>;<delay_value>
```

For numeric values, multiple numeric variables can be used:

```
ENTER 707;Result1,Result2
```

Instrument Status

Status registers track the current status of the instrument. By checking the instrument status, you can find out whether an operation has been completed, whether the instrument is receiving triggers, and more. Refer to chapter 14, “Summary Subsystem,” for for more information.

DIGitize Command

The ACQUIRE and WAVEFORM subsystems are subsystems that affect the DIGITIZE command. The DIGITIZE command is used to capture a waveform in a known format which is specified by the ACQUIRE subsystem. When the DIGITIZE command is sent to an instrument, the specified channel signal is digitized with the current ACQUIRE parameters. To obtain waveform data, you must specify the WAVEFORM parameters for the waveform data prior to sending the :WAVEFORM:DATA? query.

The number of data points comprising a waveform varies according to the number requested in the ACQUIRE subsystem. The ACQUIRE subsystem determines the number of data points, type of acquisition, and number of averages used by the DIGITIZE command. This allows you to specify exactly what the digitized information will contain. A typical setup is:

```
OUTPUT 707;":ACQUIRE:TYPE AVERAGE"<terminator>
OUTPUT 707;":ACQUIRE:COMPLETE 100"<terminator>
OUTPUT 707;":WAVEFORM:SOURCE CHANNEL1"<terminator>
OUTPUT 707;":WAVEFORM:FORMAT WORD"<terminator>
OUTPUT 707;":ACQUIRE:COUNT 4"<terminator>
OUTPUT 707;":ACQUIRE:POINTS 500"<terminator>
OUTPUT 707;":DIGITIZE CHANNEL1"<terminator>
OUTPUT 707;":WAVEFORM:DATA?"<terminator>
```

This setup places the instrument into the average mode with four averages and defines the data record to be 500 points. This means that when the DIGITIZE command is received, the waveform will not be stored into memory until 500 points have been averaged at least four times.

After receiving the :WAVEFORM:DATA? query, the instrument will start passing the waveform information when addressed to talk.

Digitized waveforms are passed from the instrument to the controller by sending a numerical representation of each digitized point. The format of the numerical representation is controlled with the :WAVEFORM:FORMAT command and may be selected as WORD, BYTE, or COMPRESSED.

The easiest method of entering a digitized waveform from the instrument is to use the WORD format and place the information in an integer array. The data point is represented by signed sixteen bit integers whose values range from 0 to 32,640. You must scale the integers to determine the voltage value of each point. These integers are passed starting with the leftmost point of the active waveform. For more information, refer to the chapter “Waveform Subsystem.”

3

Interface Functions

This section describes the interface functions and some general concepts of the HP-IB. In general, these functions are defined by IEEE 488.2. They deal with general bus management issues, as well as messages which can be sent over the bus as bus commands.

Interface Capabilities

The interface capabilities of the HP 70703A, as defined by IEEE 488.2 are SH1, AH1, T5, L4, SR1, RL1, PP1, DC1, DT1, C0, and E2.

Command and Data Concepts

The HP-IB has two modes of operation: command mode and data mode. The bus is in command mode when the ATN line is true. The command mode is used to send talk and listen addresses and various bus commands, such as a group execute trigger (GET). The bus is in the data mode when the ATN line is false. The data mode is used to convey device-dependent messages across the bus. The device-dependent messages include all of the instrument command and responses found in Chapters 5 through 18 of this manual.

Addressing

The instrument is always in addressed (talk/listen) mode. Addressed mode is used when the instrument will operate in conjunction with a controller. When the instrument is in the addressed mode, the following is true:

- Each device on the HP-IB resides at a particular address, ranging from 0 to 30.
- The active controller specifies which devices will talk, and which will listen.
- An instrument, therefore, may be talk addressed, listen addressed, or unaddressed by the controller.

If the controller addresses the instrument to talk, it will remain configured to talk until it receives an interface clear message (IFC), another instrument's talk address (OTA), its own listen address (MLA), or a universal untalk command (UNT).

If the controller addresses the instrument to listen, it will remain configured to listen until it receives an interface clear message (IFC), its own talk address (MTA), or a universal unlisten command (UNL).

Bus Commands

The following commands are IEEE 488.2 bus commands (ATN true). IEEE 488.2 defines many of the actions which are taken when these commands are received by the instrument.

Device Clear (DCL)

The device clear (DCL) or selected device clear (SDC) commands clear the input and output buffers, reset the parser, and clear any pending commands.

Group Execute Trigger (GET)

The group execute trigger (GET) command arms the trigger which is the same action produced by sending the RUN command.

Interface Clear (IFC)

The interface clear (IFC) command halts all bus activity. This includes unaddressing all listeners and the talker, disabling serial poll on all devices, and returning control to the system controller.

LED Indicators

STATUS ACT	indicates that the HP 70703A digitizing oscilloscope is active. The ACTIVE LED lights when: ■ the keyboard of the display is allocated to the oscilloscope. ■ any Display function indicates the oscilloscope (for example, when the cursor of the Display Address Map is at the HP-MSIB address of the oscilloscope). ■ an oscilloscope is a slave to another oscilloscope that is designated as a master module, and it is being used by the master oscilloscope.
STAUS ERR	indicates errors.
HP-IB RMT	indicates that the module is being remotely controlled and local control is disabled.
HP-IB LSN	indicates a state in which the module is ready to accept information from the controller.
HP-IB TLK	indicates a state in which the module is ready to send information to the controller.
HP-IB SRQ	indicates a condition requested or set by the user (for example, operation complete status, power-on condition).

4

Programming and Documentation Conventions

This section covers conventions which are used in programming the instrument, as well as conventions used in the remainder of this manual. This chapter contains a detailed description of the command tree and command tree traversal. For more information on command syntax, refer to the appendix "Message Communication and System Functions."

Truncation Rules

The truncation rule for the mnemonics used in headers and alpha arguments is: The mnemonic is the first four characters of the keyword unless:

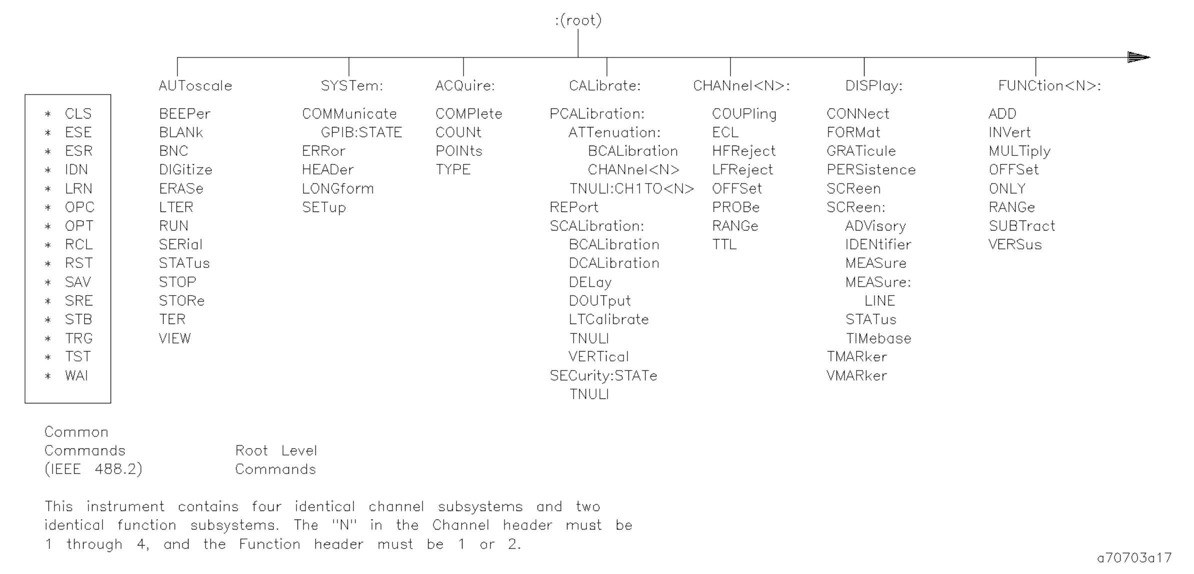
The fourth character is a vowel, then the mnemonic is the first three characters of the keyword.

This rule will not be used if the length of the keyword is exactly four characters.

Some examples of how the truncation rule is applied to various commands are shown in table 4-1.

Table 4-1. Mnemonic Truncation

Longform	**Shortform**
RANGE	RANGE
PATTERN	PATT
TIME	TIME
DELAY	DEL



This instrument contains four identical channel subsystems and two identical function subsystems. The "N" in the Channel header must be 1 through 4, and the Function header must be 1 or 2.

Figure 4-1. The HP 70703A Command Tree

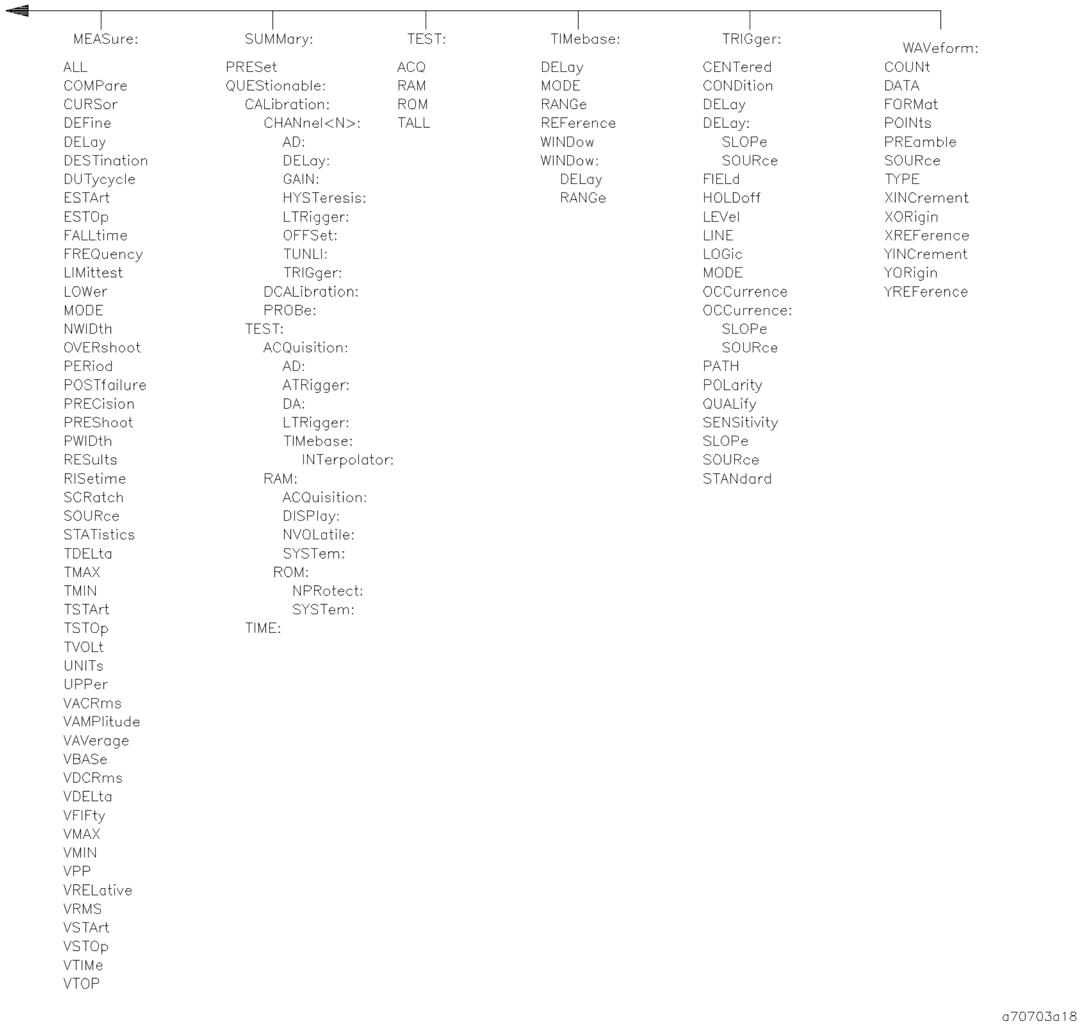


Figure 4-2. The HP 70703A Command Tree (continued)

Command Tree

The command tree (figure 4-1) shows all commands in the HP 70703A and the relationship of the commands to each other. The IEEE 488.2 common commands are not listed as part of the command tree since they do not affect the position of the parser within the tree. After a <NL> (linefeed ASCII decimal 10) has been sent to the instrument, the parser will be set to the "root" of the command tree.

Command Types

The commands for this instrument can be placed into three types:

- Common Commands
- Root Level Commands
- Subsystem Commands

Common Commands

Common commands are independent of the tree, and do not affect the position of the parser within the tree. These differ from root level commands in that root level commands place the parser back at the root, such as *RST.

Root Level Commands

The root level commands reside at the root of the command tree. These commands are always parsable if they occur at the beginning of a program message, or are preceded by a colon, such as :AUTOSCALE.

Subsystem Commands

Subsystem commands are grouped together under a common node of the tree, such as the TIMEBASE commands.

Tree Traversal Rules

Command headers are created by traversing down the command tree. A legal command header from the command tree in figure 4-1 would be ":CHANNEL1:RANGE". This is referred to as a compound header. A compound header is a header made of two or more mnemonics separated by colons. The mnemonic created contains no spaces. The following rules apply to traversing the tree:

- A leading colon or a <program message terminator> (either a <NL> or EOI true on the last byte) places the parser at the root of the command tree. A leading colon is a colon that is the first character of a program header.
- Executing a subsystem command places you in that subsystem (until a leading colon or a <program message terminator> is found). In the Command Tree, figure 4-1, use the last mnemonic in the compound header as a reference point (for example, RANGE). Then find the last colon above that mnemonic (CHANNEL1:), and that is where the parser will be. Any command below that point can be sent within the current program message without sending the mnemonic(s) which appear above them (OFFSET).

Examples

The OUTPUT statements are written using HP BASIC 5.0 on a HP 9000 Series 200/300 controller. The quoted string is placed on the bus, followed by a carriage return and linefeed (CRLF).

Example 1

```
OUTPUT 707;":CHANNEL1:RANGE 0.5;OFFSET 0"
```

Comments

The colon between CHANNEL1 and RANGE is necessary, CHANNEL1:RANGE is a compound command. The semicolon between the RANGE command and the OFFSET command is the required <program message unit separator>.

The OFFSET command does not need CHANNEL1 preceding it, since the CHANNEL1:RANGE command set the parser to the CHANNEL1 node in the tree.

Example 2

```
OUTPUT 707;":TIMEBASE:REFERENCE CENTER;DELAY 0.00001"
```

or

```
OUTPUT 707;":TIMEBASE:REFERENCE CENTER"
OUTPUT 707;":TIMEBASE:DELAY 0.00001"
```

Comments

In the first line of Example 2, the "subsystem selector" is implied for the DELAY command in the compound command.

The DELAY command must be in the same program message as the REFERENCE command, because the <program message terminator> will place the parser back at the root of the command tree.

A second way to send these commands is by placing "TIMEBASE:" before the DELAY command as shown in Example 2.

Example 3

```
OUTPUT 707;":TIM:REF CENTER;:CHAN1:OFFSET 0"
```

Comments

The leading colon before CHAN1 tells the parser to go back to the root of the command tree. The parser can then see the CHAN1:OFFSET command.

Infinity Representation

The representation of infinity is 9.99999E+37. This is also the value returned when a measurement cannot be made.

Sequential and Overlapped Commands.

IEEE 488.2 makes the distinction between sequential and overlapped commands. Sequential commands finish their task before the execution of the next command starts. Overlapped commands run concurrently, and therefore the command following an overlapped command may be started before the overlapped command is completed. All the commands of the HP 70703A are sequential.

Response Generation

IEEE 488.2 defines two times at which query responses may be buffered. The first is when the query is parsed by the instrument, the second is when the controller addresses the instrument to talk so that it may read the response. The HP 70703A will buffer responses to a query when the query is parsed.

Notation Conventions and Definitions

The following conventions are used in this manual in descriptions of remote (HP-IB or MSIB) operation:

<>	Angular brackets enclose words or characters that are used to symbolize a program code parameter or an HP-IB or MSIB command.
: : =	"is defined as." For example, <A> : : = <B> indicates that <A> can be replaced by <B> in any statement containing <A>.
|	"or." Indicates a choice of one element from a list. For example, <A> | <B> indicates <A> or <B> 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.
{ }	When several items are enclosed by braces, one, and only one of these elements must be selected.

The following definitions are used:

d: : = A single ASCII numeric character, 0-9.

n : : = A single ASCII non-zero, numeric character, 1-9.

<NL> : : = Newline or Linefeed (ASCII decimal 10).

<sp> : : = <white space>

<white space> ::= 0 through 32 (decimal) except linefeed (decimal 10).

Syntax Diagrams

At the beginning of each of the following chapters are syntax diagrams showing the proper syntax for each command. All characters contained in a circle or oblong are literals, and must be entered exactly as shown. Words and phrases contained in rectangles are names of items used with the command and are described in the accompanying text of each command. Each line can only be entered from one direction as indicated by the arrow on the entry line. Any combination of commands and arguments that can be generated by following the lines in the proper direction is syntactically correct. An argument is optional if there is a path around it. Where there is a rectangle which contains the word "space" a white space character must be entered. White space is optional in many other places.

Command Structure

The HP 70703A programming commands are divided into three types: common commands, root level commands, and subsystem commands. A programming command tree is shown in figure 4-1.

Common Commands

The common commands are the commands defined by IEEE 488.2. These commands control some functions that are common to all IEEE 488.2 instruments. Sending the common commands do not take the instrument out of a selected subsystem.

Root Level Commands

The root level commands control many of the basic functions of the instrument.

Subsystem Commands

There are several subsystems in this instrument. Only one subsystem may be selected at any given time. At power on, the command parser is set to the root of the command tree, therefore no subsystem is selected.

Note When a program message terminator or a leading colon (:) is sent in a program message, the command parser is returned to the root of the command tree.

The 12 subsystems in the HP 70703A are:

System controls some basic functions of the oscilloscope.

Acquire sets the parameters for acquiring and storing data.

Calibrate controls the internal calibrations.

Channel controls all Y-axis oscilloscope functions.

Display controls how waveforms, voltage and time markers, graticule, and text are displayed and written on the screen.

Function controls the waveform math functions of the oscilloscope.

Measure selects the automatic measurements to be made.

Summary allows monitoring of the status of the oscilloscope calibration and self test results.

Test controls the internal diagnostics.

Timebase controls all X-axis oscilloscope functions.

Trigger controls the trigger modes and parameters for each trigger mode.

Waveform provides access to waveform data, including active data from channels and functions as well as static data from waveform memories.

Program Examples

The program examples given for each command in the following chapters and appendices were written on an HP 9000 Series 200/300 controller using the HP BASIC 5.0 programming language. The programs always assume the oscilloscope is at address 707. If a printer is used, it is always assumed to be at address 701.

In these examples, special attention should be paid to the ways in which the command/query can be sent. The way the instrument is set up to respond to a command/query has no bearing on how you send the command/query. That is, the command/query can be sent using the longform or shortform if one exists for that command. You can send the command/query using upper case (capital) letters or lower case (small) letters, both work the same. Also, the data can be sent using almost any form you wish. If you were sending a channel 1 range value of 100 mV, that value could be sent using a decimal (.1), or an exponential (1e-1 or 1.0E-1), or a suffix (100 mV or 100MV).

As an example, set channel 1 range to 100 mV by sending one of the following:

- commands in longform and using the decimal format.
  OUTPUT 707;":CHANNEL1:RANGE .1"
- commands in shortform and using an exponential format.
  OUTPUT 707;":CHAN1:RANG 1E-1"
- commands using lower case letters, shortforms, and a suffix.
  OUTPUT 707;":chan1:rang 100 mV"

Note In these examples, the colon as the first character of the command is optional. The space between RANGE and the argument is required.

Command Set Organization

The command set for the HP 70703A is divided into 14 separate groups: common commands, root level commands, and 12 sets of subsystem commands. Each of the 14 groups of commands is described in the following chapters. Each of the chapters contain a brief description of the subsystem, a set of syntax diagrams for those commands, and finally, the commands for that subsystem in alphabetic order. The commands are shown in the longform and shortform using upper and lowercase letters. As an example, AUToscale indicates that the longform of the command is AUTOSCALE and the shortform of the command is AUT. Each command listing contains a description of the command and its arguments, the command syntax, and a programming example.

Table 4-2 lists the commands for the HP 70703A in alphabetical order with their corresponding subsystem or command type.

Table 4-2. Alphabetic Command Cross-Reference

Command	Where Used
ACQ	TEST Subsystem
ACQuistion	SUMMary Subsystem
AD	SUMMary Subsystem
ADD	FUNCtion Subsystem
ADVisory	DISPlay Subsystem
ALL	MEASure Subsystem
ATRigger	SUMMary Subsystem
ATTenuation	CALibrate Subsystem
AUToscale	Root Level Command
BCALibration	CALibrate Subsystem
BEEPer	Root Level Command
BLANk	Root Level Command
BNC	Root Level Command
CALibration	SUMMary Subsystem
CENTered	TRIGger Subsystem
CH1TO<N>	CALibrate Subsystem
CHANnel<N>	CALibrate Subsystem
CHANnel<N>	SUMMary Subsystem
*CLS	Common Command
COMMunicate	SYSTem Subsystem
COMPare	MEASure Subsystem
COMPlete	ACQuire Subsystem
CONDition	TRIGger Subsystem
CONNect	DISPlay Subsystem
COUNt	ACQuire Subsystem
COUNt	WAVeform Subsystem
COUPling	CHANnel Subsystem
CURSor	MEASure Subsystem
DA	SUMMary Subsystem
DATA	WAVeform Subsystem
DCALibration	CALibrate Subsystem
DCALibration	SUMMary Subsystem
DEFine	MEASure Subsystem
DELay	CALibrate Subsystem
DELay	MEASure Subsystem
DELay	SUMMary Subsystem
DELay	TIMebase Subsystem
DELay	TRIGger Subsystem
DELay:SLOPe	TRIGger Subsystem
DELay:SOURce	TRIGger Subsystem
DESTination	MEASure Subsystem
DIGitize	Root Level Command
DISPlay	SUMMary Subsystem
DOUTput	CALibrate Subsystem
DUTycycle	MEASure Subsystem
ECL	CHANnel Subsystem
ERASe	Root Level Command
ERRor	SYSTem Subsystem
*ESE	Common Command
*ESR	Common Command
ESTArt	MEASure Subsystem
ESTOp	MEASure Subsystem
FALLtime	MEASure Subsystem
FIELd	TRIGger Subsystem
FORMat	DISPlay Subsystem
FORMat	WAVeform Subsystem
FREQuency	MEASure Subsystem
GAIN	SUMMary Subsystem
GPIB:STATe	SYSTem Subsystem
GRATicule	DISPlay Subsystem

Table 4-2. Alphabetic Command Cross-Reference (continued)

Command	Where Used
HEADer	SYSTem Subsystem
HFReject	CHANnel Subsystem
HOLDoff	TRIGger Subsystem
HYSTeresis	SUMMary Subsystem
IDENtifier	DISPlay Subsystem
*IDN	Common Command
INTerpolator	SUMMary System
INVert	FUNCtion Subsystem
LEVel	TRIGger Subsystem
LFReject	CHANnel Subsystem
LIMitest	MEASure Subsystem
LINE	DISPlay Subsystem
LINE	TRIGger Subsystem
LOGic	TRIGger Subsystem
LONGform	SYSTem Subsystem
LOWer	MEASure Subsystem
*LRN	Common Command
LTCalibrate	CALibrate Subsystem
LTER	Root Level Command
LTRigger	SUMMary Subsystem
MEASure	DISPlay Subsystem
MODE	MEASure Subsystem
MODE	TIMebase Subsystem
MODE	TRIGger Susystem
MULTiply	FUNCtion Subsystem
NPRotect	SUMMary Subsystem
NVOLatile	SUMMary Subsystem
NWIDTH	MEASure Subsystem
OCCurance	TRIGger Subsystem
OCCurance:SLOPe	TRIGger Subsystem
OCCurance:SOURce	TRIGger Subsystem
OFFSet	CHANnel Subsystem
OFFSet	FUNCtion Subsystem
OFFSet	SUMMary Subsystem
ONLY	FUNCtion Subsystem
*OPC	Common Command
*OPT	Common Command
OVERshoot	MEASure Subsystem
PATH	TRIGger Subsystem
PCALibration	Calibrate Subsystem
PERiod	MEASure Subsystem
PERSistance	DISPlay Subsystem
POINts	ACQuire Subsystem
POINts	WAVeform Subsystem
POLarity	TRIGger Subsystem
POSTfailure	MEASure Subsystem
PREamble	WAVeform Subsystem
PRECision	MEASure Subsystem
PRESet	SUMMary Subsystem
PREShoot	MEASure Subsystem
PROBe	CHANnel Subsystem
PROBe	SUMMary Subsystem
PWIDth	MEASure Subsystem
QUALify	TRIGger Subsystem
QUEStionable	SUMMary Subsystem
RAM	SUMMary Subsystem
RAM	TEST Subsystem
RANGe	CHANnel Susbsytem
RANGe	FUNCtion Subsystem
RANGe	TIMebase Subsystem
*RCL	Common Command
REFerence	TIMebase Subsystem
REPort	CALibrate Subsystem
RESults	MEASure Subsystem
RISetime	MEASure Subsystem
ROM	SUMMary Subsystem
ROM	TEST Subsystem
*RST	Common Command
RUN	Root Level Command
*SAV	Common Command
SCALibration	CALibrate Subsystem
SCRatch	MEASure Subsystem
SCReen	DISPlay Subsystem
SECurity:STATe	CALibrate Subsystem
SENSitivity	TRIGger Subsystem
SERial	Root Level Command
SETup	SYSTem Subsystem
SLOPe	TRIGger Subsystem
SOURce	MEASure Subsystem
SOURce	TRIGger Subsystem

Table 4-2. Alphabetic Command Cross-Reference (continued)

Command	Where Used
SOURce	WAVeform Subsystem
*SRE	Common Command
STANdard	TRIGger Subsystem
STATistics	MEASure Subsystem
STATus	DISPlay Subsystem
STATus	Root Level Command
*STB	Common Command
STOP	Root Level Command
STORe	Root Level Command
SUBTract	FUNCtion Subsystem
SYSTem	SUMMary Subsystem
TALL	TEST Subsystem
TDELta	MEASure Subsystem
TER	Root Level Command
TEST	SUMMary Subsystem
TIME	SUMMary Subsystem
TIMebase	DISPlay Subsystem
TIMebase	SUMMary Subsystem
TMARker	DISPlay Subsystem
TMAX	MEASure Subsystem
TMIN	MEASure Subsystem
TNULl	CALibrate Subsystem
TNULl	SUMMary Subsystem
TRIGger	SUMMary Subsystem
*TRG	Common Command
*TST	Common Command
TSTArt	MEASure Subsystem
TSTOp	MEASure Subsystem
TTL	CHANnel Subsystem
TVOLt	MEASure Subsystem
TYPE	ACQuire Subsystem
TYPE	WAVeform Subsystem
UNITs	MEASure Subsystem
UPPer	MEASure Subsystem
VACRms	MEASure Subsystem
VAMPlitude	MEASure Subsystem
VAVerage	MEASure Subsystem
VBASe	MEASure Subsystem
VDCRms	MEASure Subsystem
VDELta	MEASure Subsystem
VERSus	FUNCtion Subsystem
VERTical	CALibrate Subsystem
VFIFty	MEASure Subsystem
VIEW	Root Level Command
VMARker	DISPlay Subsystem
VMAX	MEASure Subsystem
VMIN	MEASure Subsystem
VPP	MEASure Subsystem
VRELative	MEASure Subsystem
VRMS	MEASure Subsystem
VSTArt	MEASure Subsystem
VSTOp	MEASure Subsystem
VTIMe	MEASure Subsystem
VTOP	MEASure Subsystem
*WAI	Common Command
WINDow	TIMebase Subsystem
WINDow:DELay	TIMebase Subsystem
WINDow:RANGe	TIMebase Subsystem
XINCrement	WAVeform Subsystem
XORigin	WAVeform Subsystem
XREFerence	WAVeform Subsystem
YINCrement	WAVeform Subsystem
YORigin	WAVeform Subsystem
YREFerence	WAVeform Subsystem

5

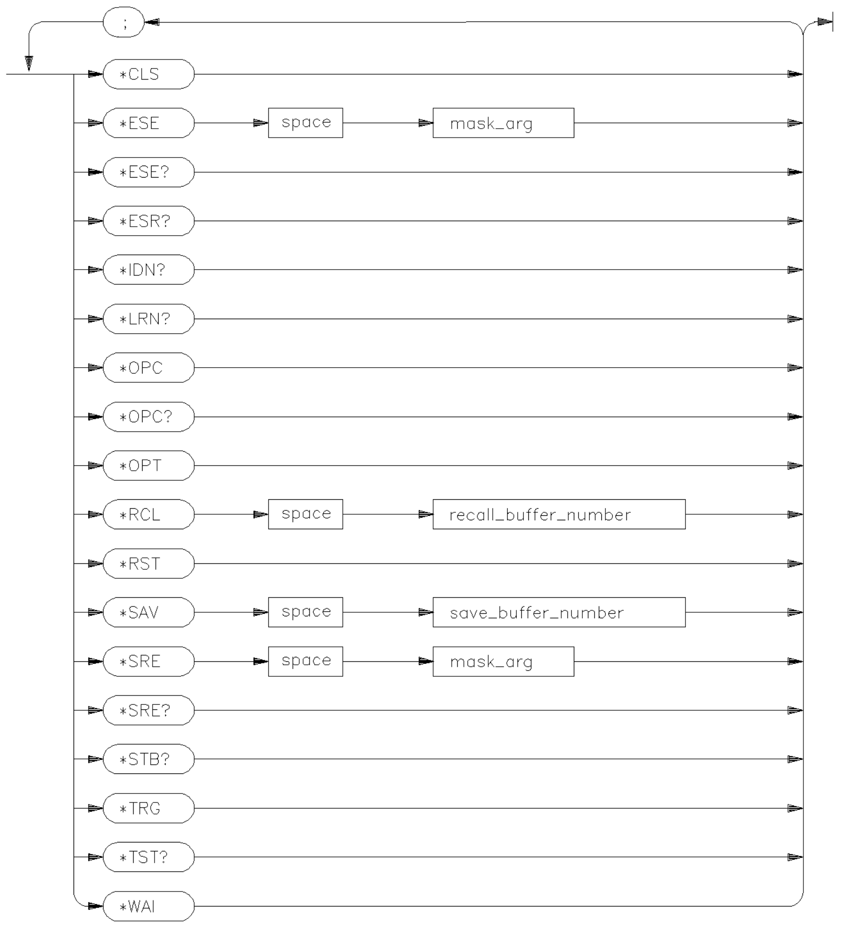
Common Commands

The Common commands are defined by the IEEE 488.2 standard. These commands will be common to all instruments that comply with this standard. They control some of the basic instrument functions, such as instrument identification and reset, reading the learn (instrument setup) string, how status is read and cleared, and how commands and queries are received and processed by the instrument.

Common commands can be received and processed by the HP 70703A whether they are sent over the HP-IB or MSIB as separate program messages or within other program messages. If an instrument subsystem has been selected and a common command is received by the instrument, the instrument will remain in the selected subsystem. For example, if the program message "ACQUIRE:COUNT 1024; *CLS; TYPE AVERAGE" is received by the instrument, the instrument will set the acquire count and type, and clear the status information. This would not be the case if some other type of command were received within the program message. For example, the program message ":ACQUIRE:COUNT 1024; :AUTOSCALE; :ACQUIRE:TYPE AVERAGE" would set the acquire count, complete the autoscale, then set the acquire type. In this example, :ACQUIRE must be sent again in order to reenter the ACQUIRE subsystem and set the type.

Refer to figure 5-1 for Common commands syntax diagram.

Note Each of the status registers mentioned in this chapter has an enable (mask) register. By setting the bits in the enable register you can select the status information you wish to use. For further information on how to read the status registers and how to use the status information available from this instrument, refer to chapter 14, "SUMMARY Subsystem."



mask_arg	=	integer, 0 through 255 This number is the sum of all the bits in the mask corresponding to conditions that are enabled. Refer to the *ESE and *SRE commands for bit definitions in the enable registers.
recall_buffer_number	=	integer, 0 through 4
save_buffer_number	=	integer, 1 through 4

a70703a19

Figure 5-1. Common Commands Syntax Diagram

*CLS
Clear Status

The *CLS common command clears the status data structures, including the device defined error queue. This command also clears the Request-for-OPC flag.

If the *CLS command immediately follows a PROGRAM MESSAGE TERMINATOR, the output queue and the MAV (message available) bit will be cleared.

Command Syntax

```
*CLS
```

Example

```
OUTPUT 707;"*CLS"
```

*ESE
Event Status Enable

The *ESE command sets the Standard Event Status Enable Register bits. The Standard Event Status Enable Register contains a mask value for the bits to be enabled in the Standard Event Status Register. A one in the Standard Event Status Enable Register will enable the corresponding bit in the Standard Event Status Register, a zero will disable the bit. Refer to table 5-1 for information about the Standard Event Status Enable Register bits, bit weights, and what each bit masks.

The *ESE query returns the current contents of the register.

Command Syntax

```
*ESE <mask>
```

Where:

```
<mask> ::= 0 to 255
```

Example

```
OUTPUT 707;"*ESE  8"
```

In this example, the *ESE 8 command will enable DDE (device dependent errors) bit 3 of the Standard Event Status Enable Register.

Query Syntax

```
*ESE?
```

Returned Format

```
<mask><NL>
```

Where:

```
<mask> ::= 0 to 255 (integer - NR1 format)
```

Example

```
OUTPUT 707;"*ESE?"
ENTER 707;Event
PRINT Event
```

Table 5-1. Standard Event Status Enable Register

Event Status Enable Register (High - Enables the ESR bit)		
Bit	**Weight**	**Enables**
7	128	PON - Power On
6	64	Not Used
5	32	CME - Command Error
4	16	EXE - Execution Error
3	8	DDE - Device Dependent Error
2	4	QYE - Query Error
1	2	Not Used
0	1	OPC - Operation Complete

*ESR
Event Status Register

The *ESR query returns the contents of the Standard Event Status Register.

Note Reading the register clears the Standard Event Status Register.

Query Syntax

```
*ESR?
```

Returned Format

```
<status><NL>
```

Where:

```
<status> ::= 0 to 255 (integer - NR1 format)
```

Example

```
OUTPUT 707;"*ESR?"
ENTER 707;Event
PRINT Event
```

Table 5-2 shows each bit in the Event Status Register and its bit weight. When you read the Event Status Register, the value returned is the total bit weights of all bits that are high at the time you read the byte.

Table 5-2. Standard Event Status Register

Bit	Bit Weight	Bit Name	Condition
7	128	PON	1 = an OFF to ON transition has occurred
6	64		0 = NOT used - always 0
5	32	CME	0 = no command errors
			1 = a command error has been detected
4	16	EXE	0 = no execution error
			1 = an execution error has been detected
3	8	DDE	0 = no device dependent errors
			1 = a device dependent error has been detected
2	4	QYE	0 = no query errors
			1 = a query error has been detected
1	2		0 = NOT used - always 0
0	1	OPC	0 = operation is not complete
			1 = operation is complete
0 = False = Low 1 = True = High			

*IDN
Identification Number

The *IDN query allows the instrument to identify itself. It returns the string:

```
"HEWLETT-PACKARD,70703A,<XXXXZYYYYY><YYMMDD>"
```

Where:

```
<XXXXZYYYYY> ::= the serial number of this instrument. The Z
                      parameter indicates the country of manufacture
                      (example, A for America, U for U.K.).

<YYMMDD> ::= the software revision of this instrument.  The first
                  two parameters (YY) represent the year, the second
                  two parameters (MM) represent the month and the final
                  two parameters (DD) represent the day of the month.
```

An *IDN query must be the last query in a message. Any queries after the *IDN query in this program message will be ignored.

Query Syntax

```
*IDN?
```

Returned Format

```
HEWLETT-PACKARD,70703A,XXXXAYYYYY,YYMMDD<NL>
```

Example

```
DIM Id$[50] OUTPUT 707;"*IDN?"
ENTER 707;Id$
PRINT Id$
```

*LRN
Learn

The *LRN query returns a program message that contains the current state of the instrument. This command allows you to store an instrument setup in the controller. The stored setup can then be returned to the instrument when you want that setup at a later time. This command performs the same function as the :SYSTEM:SETUP? query. The data can be sent to the instrument using the :SYSTEM:SETUP command.

Note The returned header for the *LRN query is :SYSTEM:SETUP.

Query Syntax

```
*LRN?
```

Returned Format

```
:SYSTem:SET <setup><NL>
```

Where:

```
<setup> ::= #800001024<learn string><NL>
```

The learn string is 1024 data bytes in length.

Example

```
DIM Lrn$[2000]
OUTPUT 707;"*LRN?"
ENTER 707 USING "-K";Lrn$
```

*OPC
Operation Complete

The *OPC command will cause the instrument to set the operation complete bit in the Standard Event Status Register when all pending device operations have finished.

The *OPC query places an ASCII “1” in the output queue when all pending device operations have finished.

Command Syntax

```
*OPC
```

Example

```
OUTPUT 707;"*OPC"
```

Query Syntax

```
*OPC?
```

Returned Format

```
1<NL>
```

Example

```
OUTPUT 707;":AUTOSCALE;*OPC?"
ENTER 707;Op$
```

*OPT
Option

The *OPT query is used to report the options installed in the instrument. The standard HP 70703A will return a zero (0). Other HP 70703A's with options will return a comma separated list of option identifiers.

Query Syntax

```
*OPT?
```

Returned Format

```
0<NL>
```

Example

```
OUTPUT 707;":AUTOSCALE;*OPT?"
ENTER 707;Opt$
```

*RCL
Recall

The *RCL command restores the state of the instrument from the specified save/recall register. An instrument setup must have been stored previously in the specified register. Registers 1 through 4 are general purpose and can be used with the *SAV command. Register 0 is special because it recalls the state that existed before the last AUTOSCALE, RECALL, ECL, or TTL operation.

Note An error message will appear on the error queue if nothing has been previously saved in the specified register.

Command Syntax

```
*RCL <rcl_register>
```

Where:

```
<rcl_register> ::= 0 through 4
```

Example

```
OUTPUT 707;"*RCL 3"<NL>
```

*RST
Reset

The *RST command places the instrument in a known state. Refer to table 5-3 for the reset conditions.

Command Syntax

```
*RST
```

Example

```
OUTPUT 707;"*RST"
```

Table 5-3. Reset Conditions for the HP 70703A

Parameter	Reset	Description
BNC	PROBe	Probe compensation ON, Trigger Out OFF.
SYSTem:		
LONGform	OFF	Character data returned in shortform.
ACQuire:		
COMPlete	100	Acquisition complete when at 100%.
COUNt	8	8 hits per time bucket for completion (will return “1” in NORMal mode).
POINts	500	Acquisition record contains 500 pts.
TYPe	NORMAL	Acquisition complete in 1 count.
CHANnel:	1	Channel 1 ON, Channels 2-4 OFF
COUPling	DC	Coupling to DC on all channels.
HFReject	OFF	Internal low pass filter OFF on all channels.
LFReject	OFF	Internal high pass filter OFF on all channels.
OFFSet	0 V	Centerscreen is 0 V on all channels.
PROBe	1:1	Probe attenuation factor is 1:1 on all channels.
RANGe	4 V	Full-scale vertical scale is 4 V.
DISPlay:		
CONNect	OFF	Connect dots on traces OFF.
FORMat	1	One trace display area.
GRATicule	AXIS	Display AXIS graticule.
PERSistence	SINGle	Persistence set to minimum.
TMARker	OFF	Time markers OFF.
VMARker	OFF	Voltage markers OFF.

Table 5-3. Reset Conditions for the HP 70703A (continued)

Parameter	Reset	Description
FUNCtion:	OFF	FUNCtion 1 and 2 OFF.
MEASure:		
DESTination	OFF	Destination function OFF.
LIMittest	OFF	Limit test function OFF.
LOWer	10	Lower measurement threshold to 10%.
MODe	STANdard	Measurement performed using IEEE standard definitions and thresholds.
POSTfailure	STOP	Limit test stopped after violation.
SOURce	CHANnel1	Measurement source to channel 1.
STATistics	OFF	Current measurement is returned.
UNITs	PERCent	Threshold units to percent.
UPPer	90	Upper measurement threshold to 90%.
TIMebase:		
DELay	0 s	Time base delay to 0 seconds.
MODe	AUTo	Time base mode set to auto trigger.
RANGe	1 ms	Full-scale horizontal time to 1 ms.
REFerence	CENTer	Delay reference set to center of sweep.
WINDow:	OFF	Second time base to OFF.
DELay	0 s	Second time base delay to 0 seconds.
RANGe	1 ms	Second time base full scale horizontal time to 1 ms.
TRIGger:		
HOLDoff	TIMe,40 ns	Holdoff set to 40 ns.
LEVel	0 V	Trigger level at 0 V.
MODe	EDGe	Edge trigger mode active.
SENSitivity	NORMal	Noise reject OFF.
SLOPe	POSitive	Positive edge trigger.
SOURce	CHANnel1	Channel 1 produces trigger.
WAVeform:		
FORMat	BYTE	Waveform data output to BYTE.
SOURce	CHANnel1	Channel 1 source for waveform commands.

*SAV
SAVE

The *SAV command stores the current state of the device in a save register. The data parameter is the number of the save register where the data will be saved. Registers 1 through 4 are valid for this command.

Command Syntax

```
*SAV <save_register>
```

Where:

```
<save_register> ::= 1 through 4
```

Example

```
OUTPUT 707;"*SAV 3"
```

*SRE
Service Request Enable

The *SRE command sets the Service Request Enable Register bits. The Service Request Enable Register contains a mask value for the bits to be enabled in the Status Byte Register. A one in the Service Request Enable Register will enable the corresponding bit in the Status Byte Register, a zero will disable the bit. Refer to table 5-4 for the bits in the Service Request Enable Register and what they mask. The *SRE query returns the current value.

Command Syntax

```
*SRE <mask>
```

Where:

```
<mask> ::= 0 to 255
```

Example

```
OUTPUT 707;"*SRE 16"
```

Note This example enables a service request to be generated when a message is available in the output queue. When a message is available the MAV bit will be high.

Query Syntax

```
*SRE?
```

Returned Format

```
<mask><NL>
```

Where:

```
<mask> ::= sum of all bits that are set - 0 through 255
                       (integer - NR1 format)
```

Example

```
OUTPUT 707;"*SRE?"
ENTER 707;Value
PRINT Value
```

Table 5-4. Service Request Enable Register

Service Request Enable Register (High - Enables the SRE bit)		
Bit	**Weight**	**Enables**
7	128	not used
6	64	bit ignored - always zero
5	32	ESB - Event Status Bit
4	16	MAV - Message Available
3	8	LTF - Limit Test Fail
2	4	QUE - Questionable Status
1	2	bit ignored - always zero
0	1	TRG - Trigger

*STB
Status Byte

The *STB query returns the current value of the instrument's status byte. The (Master Summary Status) bit is reported on bit 6 instead of the RQS (request service) bit. The MSS indicates whether or not the device has at least one reason for requesting service. Refer to table 5-5 for the meaning of the bits in the status byte.

Note To read the instrument's status byte with RQS reported on bit 6, use the HP-IB Serial Poll.

Query Syntax

```
*STB?
```

Returned Format

```
<value><NL>
```

Where:

```
<value> ::= 0 through 255 (integer - NR1)
```

Example

```
OUTPUT 707;"*STB?"
ENTER 707;Value
PRINT  Value
```

Table 5-5. The Status Byte Register

Bit	Bit Weight	Bit Name	Condition
7	128	—	0 = not used
6	64	RQS/MSS	0 = instrument has no reason for service
			1 = instrument is requesting service
5	32	ESB	0 = no event status conditions have occurred
			1 = an enabled event status condition has occurred
4	16	MAV	0 = no output messages are ready
			1 = an output message is ready
3	8	LTF	0 = no limit test has failed
			1 = limit test has failed
2	4	QUE	0 = no questionable status condition has occurred
			1 = a questionable status condition has occurred
1	2		0 = not used
0	1	TRG	0 = no trigger has occurred
			1 = a trigger has occurred
0 = False = Low 1 = True = High			

*TRG
Trigger

The *TRG command has the same effect as the Group Execute Trigger (GET). This effect is as if the RUN command had been sent.

Command Syntax

```
*TRG
```

Example

```
OUTPUT 707;"*TRG"
```

*TST
Test

The *TST query causes the instrument to perform a self-test. The result of the test will be placed in the output queue.

Note Prior to sending this command all front panel inputs must be disconnected.

A 0 indicates the test passed and a non-zero value indicates the test failed. If a test fails, refer to the troubleshooting section of the Service Manual.

Query Syntax

```
*TST?
```

Returned Format

```
<result><NL>
```

Where:

```
<result> ::= 0 or non-zero value
```

Where:

```
0 indicates the test passed.
Non-zero indicates the test failed.
```

Example

```
OUTPUT 707;"*TST?"
ENTER 707;Result
PRINT Result
```

*WAI
Wait

The *WAI command has no function in the HP 70703A, but is parsed for compatibility with other instruments.

Command Syntax

```
*WAI
```

Example

```
OUTPUT 707;"*WAI"
```

Root Level Commands

The Root Level commands control many of the basic operations of the oscilloscope. These commands will always be recognized by the parser if they are prefixed with a colon, regardless of current command tree position. After executing a root level command, the parser is positioned at the root of the command tree.

Figure 6-1 lists the Root Level commands syntax diagram.

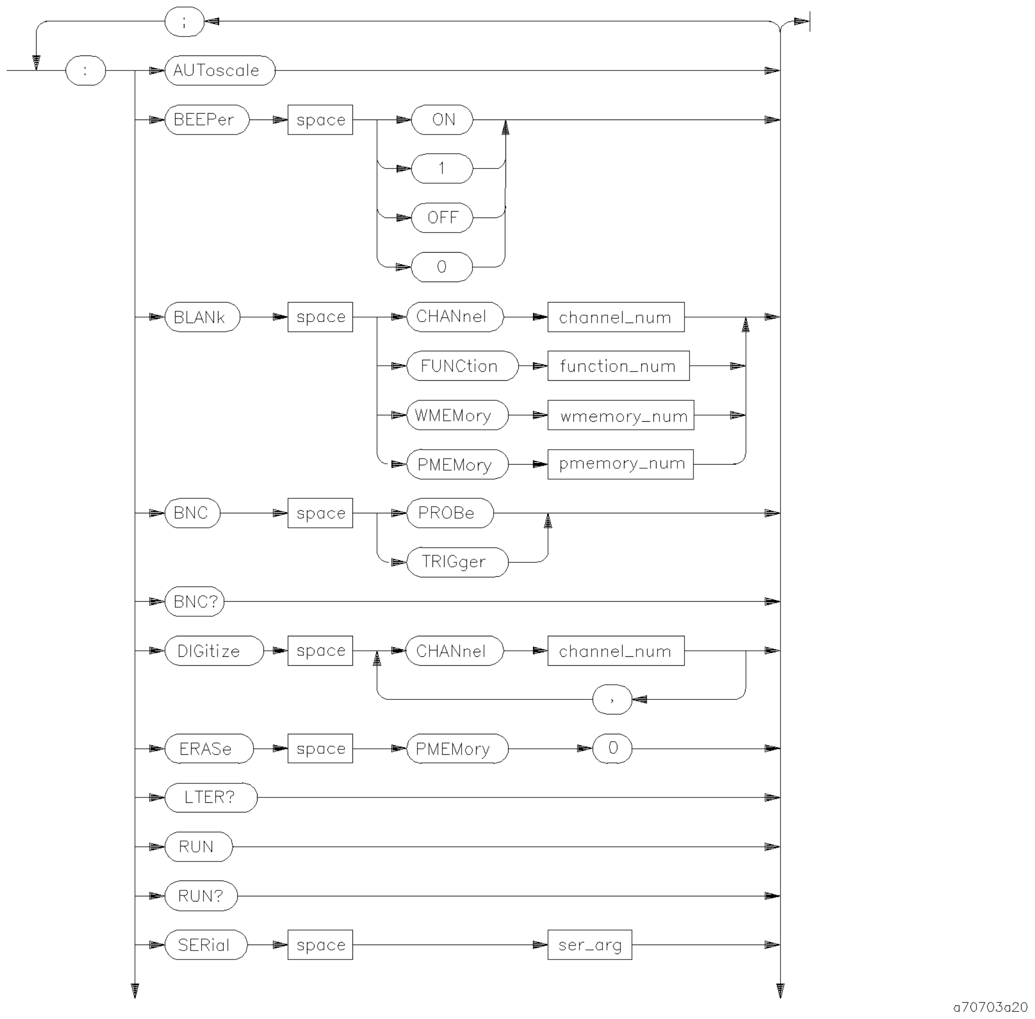


Figure 6-1. Root Level Commands Syntax Diagram

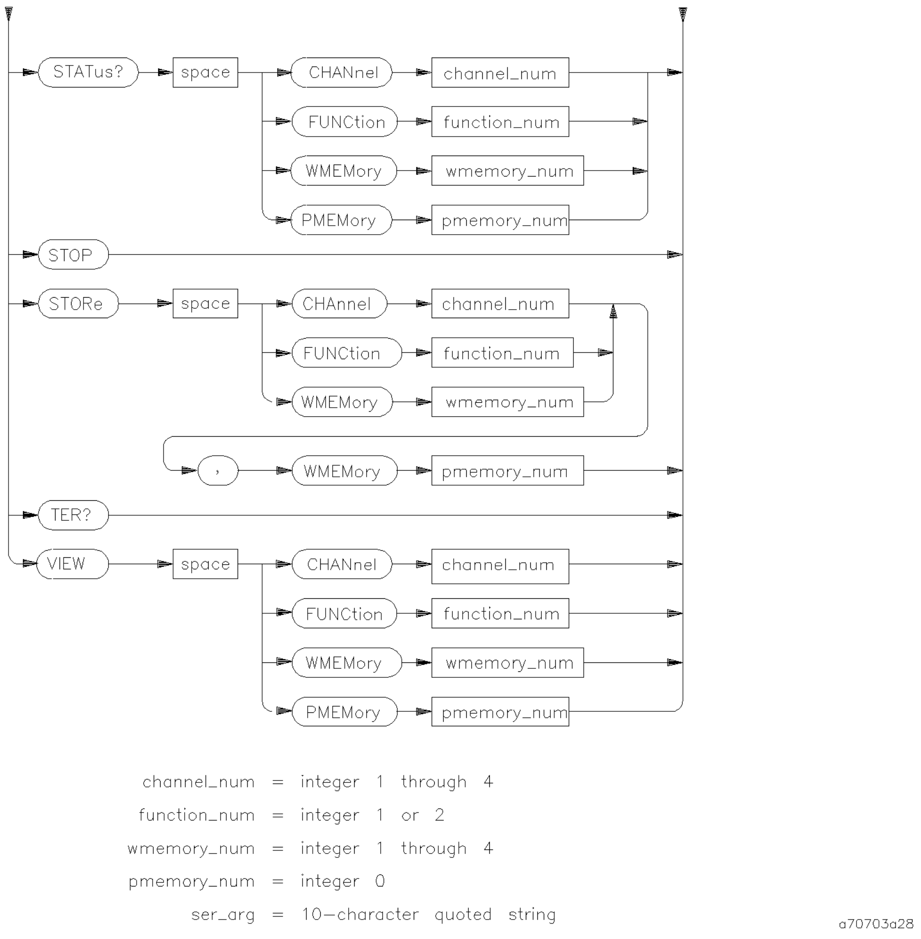


Figure 6-2. Root Level Commands Syntax Diagram (continued)

AUToscale

The AUTOSCALE command causes the oscilloscope to evaluate all input signals and set the correct conditions to present the signals. When the AUTOSCALE command is sent, the following conditions are set:

- the vertical sensitivity
- the vertical offset
- the trigger to edge mode with minimum persistence
- the trigger level, holdoff, and slope
- the timebase range as required

In addition, the AUTOSCALE command turns off:

- markers
- all measurements
- functions
- windows
- memories
- connect the dots

If signals are present on more than one input, the sweep will be triggered on the signal closest to channel 1. If a signal is not present on channel 1 then the oscilloscope will be triggered on channel 2. If a signal is not present on channel 2 then the oscilloscope will be triggered on channel 3, and so on. If no signals are found on any input, the oscilloscope is returned to its former state.

Command Syntax

```
:AUToscale
```

Example

```
OUTPUT 707;":AUTOSCALE"
```

BEEPer

The BEEPER command controls any audio beeper available to the instrument when the instrument has a keyboard link established with an MMS keyboard device.

This beeper is used by the user interface to signal certain conditions (such as errors) which need to be drawn to the users attention.

Sending the BEEPER command with no arguments will sound the beeper if it is currently available and enabled.

Command Syntax

```
:BEEPer {{ON|1}|{OFF|0}}
```

Example

```
OUTPUT 707;":BEEPER ON"      Enable beeper
OUTPUT 707;":BEEPER"         Sound beeper if available
```

Query Syntax

```
:BEEPer?
```

Returned Format

```
{1|0}<NL>
```

Where:

```
=
1 ::= ON
0 ::= OFF
```

Example

```
OUTPUT 707;"BEEPER?"
ENTER 707;State
PRINT State
```

BLANk

The BLANK command causes the instrument to turn off or stop presenting the specified channel, function, pixel memory, or waveform memory. To blank a specified channel use the command :BLANK CHANNEL{1|2|3|4}. To blank a waveform memory use :BLANK WMEMORY{1|2|3|4}, to blank a current display use the command :BLANK PMEMORY0, and to blank a function use the command :BLANK FUNCTION{1|2}.

Command Syntax

```
:BLANk <display>
```

Where:

```
<display> ::= {CHANnel{1|2|3|4}|FUNCtion{1|2}|
              WMEMory{1|2|3|4}|PMEMory0}
```

Example

```
OUTPUT 707;":BLANK CHANNEL1"
```

BNC

The BNC command selects the output mode of the Probe Compensation AC Calibrator Output BNC connector to PROBE or TRIGGER. The PROBE mode outputs a square wave signal and the TRIGGER mode outputs a rising edge when an internal trigger occurs.

The BNC query returns the current mode for the Probe Compensation AC Calibrator Output BNC connector.

Command Syntax

```
:BNC {PROBe|TRIGger}
```

Example

```
OUTPUT 707;":BNC PROBE"
```

Query Syntax

```
:BNC?
```

Returned Format

```
{PROBe|TRIGger}<NL>
```

Example

```
OUTPUT 707;":BNC?"
ENTER 707;Mode$
PRINT Mode$
```

DIGitize

The DIGITIZE command is used to acquire waveform data for transfer over the HP-IB or MSIB. It causes an acquisition to take place on the specified channel(s) with the resulting data being placed in the channel buffer.

The ACQUIRE subsystem commands are used to set up conditions such as TYPE, number of POINTS, and the COUNT for the next DIGITIZE command. See the ACQUIRE subsystem for a description of these commands. To determine the actual number of points that are acquired and how the data will be transferred, refer to the WAVEFORM Subsystem commands. For more information on the DIGITIZE command refer to the section on the DIGITIZE command in the chapter "Introduction to Programming an Instrument."

Note Sending the DIGITIZE command will turn off any unused channels.

When the digitize operation is complete, the instrument is placed in the stopped mode. When the instrument is restarted with a RUN command, the digitized data stored in the channel buffers will be overwritten. Therefore, ensure all operations that require the digitized data are completed before restarting the HP 70703A.

The speed of the total digitize operations may be improved if two or more DIGITIZE commands are sent without changing other parameters.

The sources for the :DIGITIZE command are channels 1 through 4.

Command Syntax

```
:DIGitize CHANnel<N>[,CHANnel<N>] ...
```

Where:

```
<N> ::= 1, 2, 3, or 4.
```

Example

```
OUTPUT 707;":DIGITIZE CHANNEL1,CHANNEL2"
```

ERASe

The ERASE command erases the current display.

If the scope is running and being triggered and ERASE PMEMORY0 is executed, the instrument will momentarily stop acquiring data, clear the contents of the current display, and then continue with data acquisition.

Command Syntax

```
:ERASe PMEMory0
```

Example

```
OUTPUT 707;":ERASE PMEMORY0"
```

LTER
Limit Test Event Register

The LTER query allows the Limit Test Event Register to be read. The Limit Test Event Register contains the Limit Test Fail bit. This bit is set when the limit test is active and a limit test has failed. After the Limit Test Event Register is read, it is cleared.

A Service Request (SRQ) can only be generated when the bit transitions from 0 to 1, therefore the bit must be cleared each time you would like a new Service Request to be generated.

Query Syntax

```
:LTER?
```

Returned Format

```
{1|0}<NL>
```

Example

```
OUTPUT 707;":LTER?"
ENTER 707;Lmt$
PRINT Lmt$
```

RUN

The RUN command acquires data for the active waveform. The data is acquired in the manner defined by the timebase mode.

If the timebase mode is in SINGLE, the RUN command enables the trigger once and saves the acquired data.

If the timebase mode is AUTO or TRIGGERED, the RUN command enables the trigger repeatedly and saves the data it acquires continuously. See the :TIMEBASE:MODE command for a description of the various modes. The RUN query returns the current RUN state.

Command Syntax

```
:RUN
```

Example

```
OUTPUT 707;":RUN"
```

Query Syntax

```
:RUN?
```

Returned Format

```
<state><NL>
```

Where:

```
<state> ::= "1" = RUN, "0" = STOP
```

Example

```
OUTPUT 707;"RUN?"
ENTER 707;Run_state
PRINT Run_state
```

SERial
Serial Number

The SERIAL command allows you to enter a serial number in the instrument. The instrument serial number is entered at the factory, therefore this will normally not be required. Do not use this command unless you need to serialize the instrument for a different application.

The serial number is placed in protected non-volatile ram, so the protection switch must be in the unprotected position to write a new serial number to the instrument.

This serial number is part of the string returned for the *IDN? query.

Command Syntax

```
:SERial <string>
```

Where:

```
<string> ::= 10 character serial number within quotes
```

Example

```
OUTPUT 707;":SER ""1234U56789"""
```

STATus

The STATUS query indicates whether a channel, function, wmemory, or pmemory is ON or OFF. A one indicates ON and a zero indicates OFF. PMEMORY0 indicates the current display.

Query Syntax

```
:STATus? <display>
```

Where:

```
<display> ::= {CHANnel{1|2|3|4}|FUNCtion{1|2}|WMEMory{1|2|3|4}|PMEMory0}
```

Returned Format

```
{0|1}<NL>
```

Example

```
OUTPUT 707;":STATUS? CHANNEL1"
ENTER 707;Status$
PRINT Status$
```

STOP

The STOP command causes the instrument to stop acquiring data for the active waveform.

The RUN command must be executed to restart data acquisition.

Command Syntax

```
:STOP
```

Example

```
OUTPUT 707;":STOP"
```

STORe

The STORE command moves a stored waveform, channel, or function to a waveform memory. This command has two parameters. The first is the source of the waveform. The source can be specified as any channel, function, or waveform memory. The second parameter is the destination of the waveform, which can only be waveform memory 1 through 4. The current display cannot be stored as a single item.

Command Syntax

```
:STORe <source>,<destination>
```

Where:

```
<source> ::= {CHANnel{1|2|3|4}|FUNCtion{1|2}|WMEMory{1|2|3|4}}
<destination> ::= WMEMory {1|2|3|4}
```

Example

```
OUTPUT 707;":STORE CHANNEL2,WMEMORY4"
```

TER
Trigger Event Register

The TER query allows the Trigger Event Register to be read. When the Trigger Event Register is read it is cleared. A one indicates a trigger has occurred. A zero indicates a trigger has not occurred.

If a trigger event is not found and the sweep is auto-triggering this bit will not be set.

A Service Request (SRQ) can only be generated when the bit transitions from 0 to 1, therefore the bit must be cleared each time you would like a new Service Request to be generated.

Query Syntax

```
:TER?
```

Returned Format

```
{1|0}<NL>
```

Example

```
OUTPUT 707;":TER?"
ENTER 707;Trg_event$
PRINT Trg_event$
```

VIEW

The VIEW command causes the instrument to turn on an active channel, function, display, or waveform memory.

To turn on a channel use the command :VIEW CHANnel{1|2|3|4}, for the current display, use the parameter :VIEW PMEMory0, and to turn on a function send the command :VIEW FUNCtion{1|2}.

The BLANK command causes the instrument to turn off a specified channel, function, display, or waveform memory.

Command Syntax

```
:VIEW {CHANnel{1|2|3|4}|FUNCtion{1|2}|PMEMory0|WMEMory{1|2|3|4}}
```

Example

```
OUTPUT 707;":VIEW CHANNEL1"
```

7

System Subsystem

The SYSTEM subsystem commands control the way in which query responses are formatted.

Refer to figure 7-1 for SYSTEM subsystem commands syntax diagram.

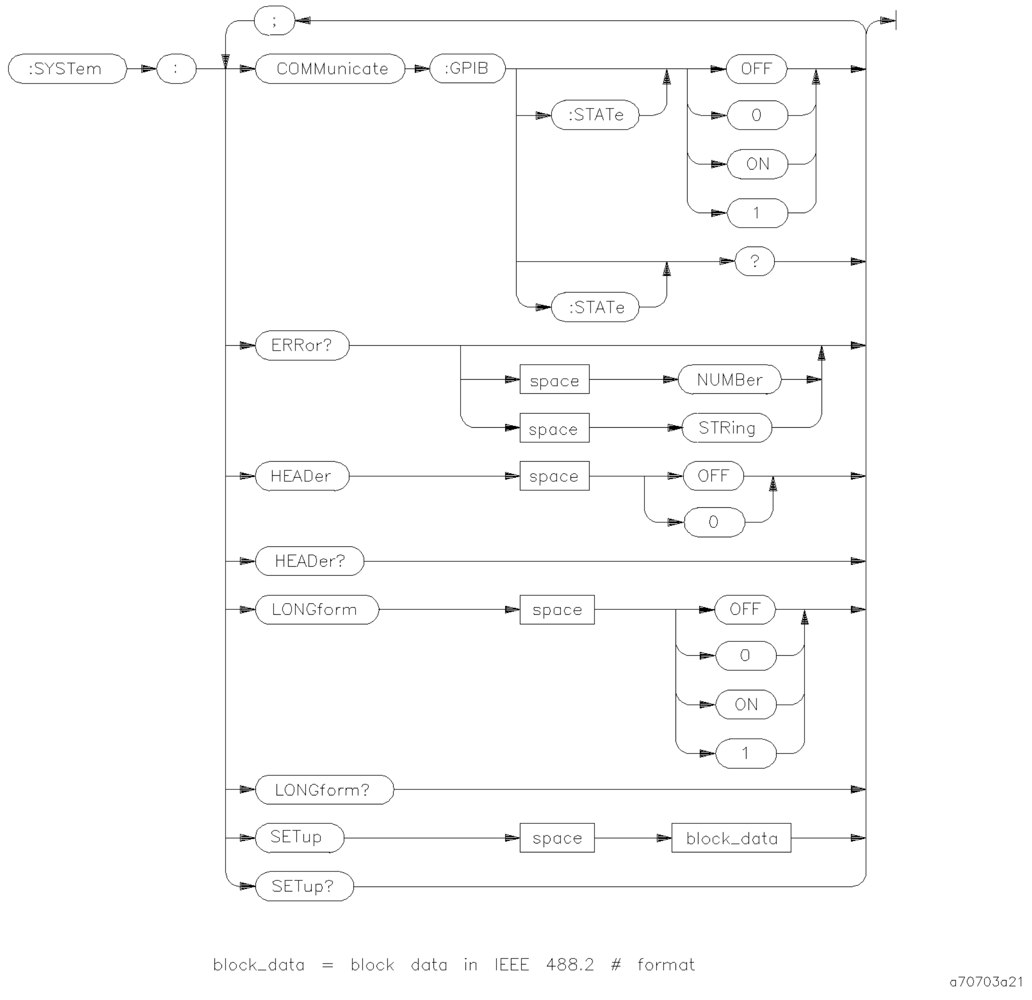


Figure 7-1. SYSTEM Subsystem Commands Syntax Diagram

COMMunicate:GPIB[:STATe]

The COMMUNICATE:GPIB[:STATE] command controls the power to the HP-IB interface. To control this instrument via HP-IB, the power to the HP-IB interface has to be enabled. However, in large systems where this instrument is controlled over MSIB, disabling the power to the HP-IB interface reduces the effective number of bus loads on the HP-IB interface by one.

Note Once HP-IB is disabled (OFF), the HP 70703A will **NOT** respond to any HP-IB activity until the HP-IB is enabled from the user interface or over MSIB. Use this command with caution.

Command Syntax

```
:SYSTem:COMMunicate:GPIB[:STATe] {{ON|1}|{OFF|0}}
```

Example

```
OUTPUT 707;":SYSTEM:COMMUNICATE:GPIB:STATE ON"
OUTPUT 707;":SYSTEM:COMMUNICATE:GPIB ON"
```

Query Syntax

```
:SYSTem:COMMunicate:GPIB[:STATe]?
```

Returned Format

```
{1|0}<NL>
```

Where:

```
1::= ON
0::= OFF
```

Example

```
OUTPUT 707;":SYSTEM:COMMUNICATE:GPIB?"
ENTER 707;State
PRINT State
```

ERRor

The :SYSTEM:ERROR query outputs the next error number in the error queue over the HP-IB or MSIB. This instrument has an error queue that is 30 errors deep and operates on a first-in, first-out basis. Successively sending the query, :SYSTEM:ERROR?, returns the error numbers in the order that they occurred until the queue is empty. Any further queries then return zeros until another error occurs.

When the NUMBER parameter is used in the query only the numeric error code is output. When the STRING parameter is used, the error number is output followed by a comma and a quoted string. If no parameter is specified, then the numeric error code is output. No parameter specified is the same as specifying NUMBER.

See table 7-1 for the error numbers.

Query Syntax

```
:SYSTem:ERRor? {NUMBer|STRing|(no_param)}
```

Returned Format

```
<error>[,<quoted string>]<NL>
```

Where:

```
<error> ::= an integer error code
<quoted string> ::= an alpha string specifying the error condition
```

Example

```
DIM Emsg$[50]
OUTPUT 707;":SYSTEM:ERROR?"
ENTER 707;Emsg$
PRINT Emsg$
```

Table 7-1. Error Messages

Error Number	Description
12	Edges required not found
70	RAM write protected
71	RAM is hardware write protected
−100	Command error
−101	Invalid character received
−102	Syntax error
−103	Invalid separator
−104	Data type error
−105	GET not allowed
−108	Parameter not allowed
−109	Missing parameter
−112	Program mnemonic too long
−113	Undefined header
−121	Invalid character in numbers
−123	Numeric overflow
−124	Too many digits
−128	Numeric data not allowed
−130	Suffix error
−131	Invalid suffix
−138	Suffix not allowed
−140	Character data error
−141	Invalid character data
−148	Character data not allowed
−150	String data error
−151	Invalid string data
−158	String data not allowed
−161	Invalid block data
−168	Block data not allowed
−170	Expression error
−171	Invalid expression
−178	Expression data not allowed
−181	Invalid outside macro definition
−183	Invalid inside macro definition

Table 7-1. Error Messages (continued)

Error Number	Description
−200	Execute error
−211	Trigger ignored
−213	Unit ignored
−221	Setting conflict
−222	Data out of range
−223	Too much data
−270	Macro error
−272	Macro execution error
−273	Illegal macro label
−276	Macro recursion error
−277	Macro redefinition not allowed
−310	System error
−314	Save/recall memory loss
−315	Configuration memory loss
−330	Self-test failed
−350	Queue overflow
−400	Query Error
−410	Query INTERRUPTED
−420	Query UNTERMINATED
−430	Query DEADLOCKED
−440	Query UNTERMINATED after indefinite response

HEADer

The :SYSTEM:HEADER command has no function in the HP 70703A but is parsed for compatibility with other instruments. Note that ON is not a valid parameter.

Command Syntax

```
:SYSTem:HEADer {OFF|0}
```

Example

```
OUTPUT 707;":SYSTEM:HEADER OFF"
```

Query Syntax

```
:SYSTem:HEADer?
```

Returned Format

```
0<NL>
```

Where:

```
0::= OFF
```

Example

```
OUTPUT 707;":SYSTEM:HEADER?"
ENTER 707;State
PRINT State
```

LONGform

The :SYSTEM:LONGFORM command sets the longform variable which tells the HP 70703A how to format query responses. If the LONGFORM command is set to OFF, alpha arguments are sent from the HP 70703A in the shortform. If the LONGFORM command is set to ON, the whole word will be output. This command does not affect the input data messages to the HP 70703A. Headers and arguments may be sent to the HP 70703A in either longform or shortform regardless of how the LONGFORM command is set.

The LONGFORM query returns the state of the LONGFORM command.

Note Even though the LONGFORM command can be sent using an alpha or numeric argument, the response is always a 1 or 0 (1 for ON, 0 for OFF).

Command Syntax

```
:SYSTem:LONGform {{ON|1}|{OFF|0}}
```

Example

```
OUTPUT 707;":SYST:LONG ON"
```

Query Syntax

```
:SYSTem:LONGform?
```

Returned Format

```
{1|0}<NL>
```

Where:

```
1 ::= ON
0 ::= OFF
```

Example

```
DIM Long$[30]
OUTPUT 707;":SYSTEM:LONGFORM?"
ENTER 707;Long$
PRINT Long$
```

SETup

The :SYSTEM:SETUP command sets the HP 70703A as defined by the data in the learn string sent from the controller. The setup string contains 1024 bytes of setup data. The 1024 bytes does not include the preamble.

The SETUP query outputs the current HP 70703A setup in the form of a learn string to the controller. The SETUP query operates the same as the *LRN? query.

The learn string is sent and received as a binary block of data. The format for the data transmission is the # format defined in the IEEE 488.2 specification.

Command Syntax

```
:SYSTem:SETup <setup>
```

Example

```
OUTPUT 707;":SYSTEM:SETUP <setup>"
```

Where:

```
<setup> ::= #800001024<setup data string>
```

Query Syntax

```
:SYSTem:SETup?
```

Returned Format

```
<setup><NL>
```

Where:

```
<setup> ::= #800001024<setup data string>
```

Note The logical order for this instruction is to send the query first followed by the command at a time of your choosing. The query causes the learn string to be sent to the controller and the command causes the learn string to be returned to the HP 70703A.

Example

```
10  DIM Set$[2000]
20  !   Setup the instrument as desired
30  OUTPUT 707;":SYST:HEAD OFF"
40  OUTPUT 707;":SYSTEM:SETUP?"
50  !   Transfer the instrument setup to controller
60  ENTER 707 USING "-K";Set$  !Store the setup
70  PAUSE  !Change the setup as desired
80  OUTPUT 707 USING "#,K";":SYST:SETUP ";Set$
90  !   Returns the instrument to the first setup
100 END
```

8

Acquire Subsystem

The ACQUIRE subsystem commands set up conditions for executing a DIGITIZE root level command to acquire waveform data. This subsystem selects the type of data, the number of averages, the number of data points and the completion criteria.

Refer to figure 8-1 for the ACQUIRE subsystem commands syntax diagram.

Note The term “Time Buckets” is defined as - the time range divided into a specific number of horizontal time points as defined by the :ACQUIRE:POINTS command. Each of these increments in time have a fixed time associated with it.

(Normal) Persistence Mode

The :ACQUIRE:TYPE NORMAL command is used for general purpose type measurements. The waveform reflects the last data point (hit) in each time bucket. ACQUIRE:COUNT has no effect in this mode. The :ACQUIRE:COUNT query will always return a 1 when the acquisition type is set to NORMAL.

Averaging Mode

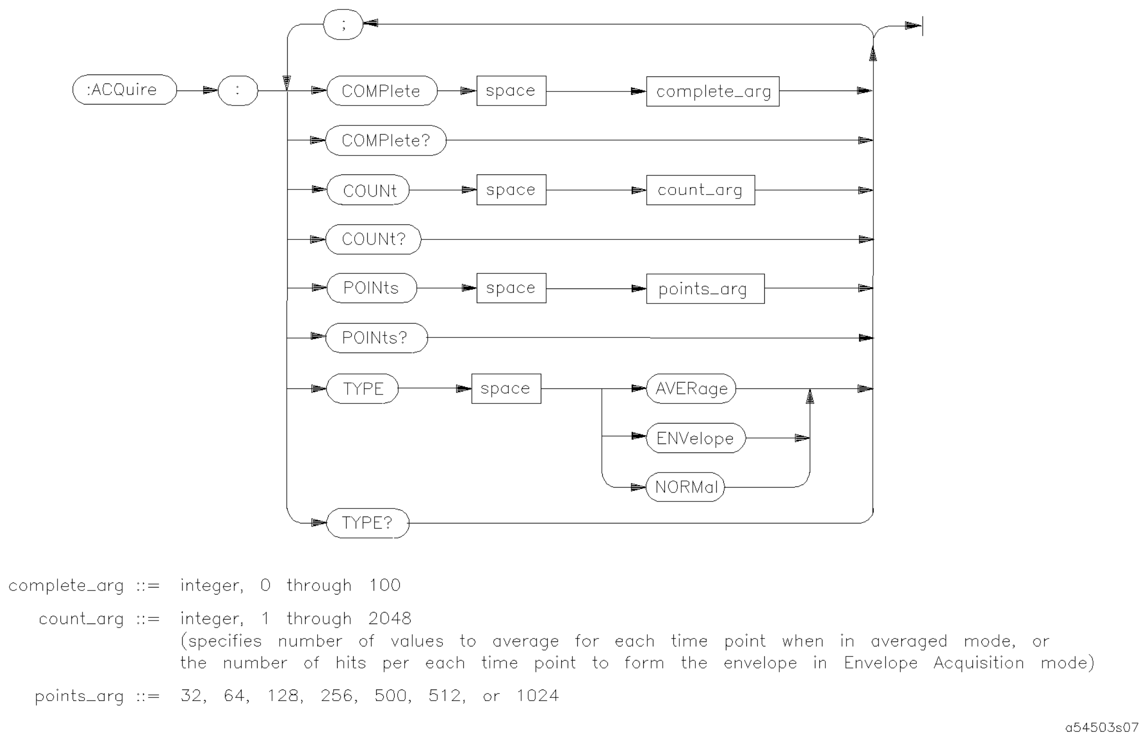
The :ACQUIRE:TYPE AVERAGE command is used when reduction of signal noise and improved resolution is desired. The waveform reflects a minimum of <N> acquisitions averaged per time bucket, where <N> is the current ACQUIRE:COUNT <N> value.

COUNT can be set in AVERAGE mode by sending the :ACQUIRE:COUNT command followed by the number of averages. In this mode the value is rounded to the nearest power of 2. It determines the number of averages that must be acquired.

Envelope Mode

The :ACQUIRE:TYPE ENVELOPE command is used when measuring voltage or time jitter. The waveform reflects the minimum and maximum data points (hit) in each time bucket.

A count value can be set in the envelope mode. This value determines the number of values to be used, at each time point, when constructing the envelope.



:ACQuire
:
;
COMPlete
space
complete_arg
COMPlete?
COUNt
space
count_arg
COUNt?
POINts
space
points_arg
POINts?
TYPE
space
AVERage
ENVelope
NORMal
TYPE?
complete_arg ::= integer, 0 through 100
count_arg ::= integer, 1 through 2048 (specifies number of values to average for each time point when in averaged mode, or the number of hits per each time point to form the envelope in Envelope Acquisition mode)
points_arg ::= 32, 64, 128, 256, 500, 512, or 1024
a54503s07

Figure 8-1. ACQUIRE Subsystem Commands Syntax Diagram

COMPlete

The :ACQUIRE:COMPLETE command specifies the completion criteria for an acquisition. The parameter determines what percentage of the time buckets need to be "full" before an acquisition is considered complete. If you are in the NORMAL mode the instrument only needs one data bit per time bucket for that time bucket to be considered full. In order for the time bucket to be considered full in the AVERAGE or ENVELOPE modes a specified number of data points (COUNT) must be acquired.

The range for the COMPLETE command is 0 to 100 and indicates the percentage of time buckets that must be "full" before the acquisition is considered complete. If the complete value is set to 100%, all time buckets must contain data for the acquisition to be considered complete.

If the complete value is set to 0, then one acquisition cycle will take place.

The COMPLETE query returns the completion criteria for the currently selected mode.

Command Syntax

```
:ACQuire:COMPlete <comp>
```

Where:

```
<comp> ::= 0 to 100 percent
```

Example

```
OUTPUT 707;":ACQUIRE:COMPLETE 85"
```

Query Syntax

```
:ACQuire:COMPlete?
```

Returned Format

```
<comp><NL>
```

Where:

```
<comp> ::= 0 to 100 (integer - NR1 format)
```

Example

```
DIM Cmp$[50]
OUTPUT 707;":ACQUIRE:COMPLETE?"
ENTER 707;Cmp$
PRINT Cmp$
```

COUNt

In average mode, :ACQUIRE:COUNT specifies the number of values to be averaged for each time bucket before the acquisition is considered complete for that time bucket.

When acquisition type is set to NORMAL, the count is 1.

When the acquisition type is set to AVERAGE, the count can range from 1 to 2048. Any value can be sent, however the value will be rounded to the nearest power of 2.

When the acquisition type is set to ENVELOPE, the count can be any value between 1 and 2048.

The COUNT query returns the currently selected count value.

Command Syntax

```
:ACQuire:COUNt <count>
```

Where:

```
<count> ::= 1 to 2048
```

Example

```
OUTPUT 707;":ACQUIRE:TYPE AVERAGE;COUNT 1024"
```

Query Syntax

```
:ACQuire:COUNt?
```

Returned Format

```
<count><NL>
```

Where:

```
<count> ::= 1 through 2048 (integer - NR1 format)
```

Example

```
DIM Cnt$[50]
OUTPUT 707;":ACQ:COUNT?"
ENTER 707;Cnt$
PRINT Cnt$
```

POINts

The :ACQUIRE:POINTS command specifies the number of time buckets for each acquisition record. The legal settings are 32, 64, 128, 256, 500, 512, or 1024. Any value between 32 and 1024 can be sent to the instrument. If a value is sent that is not one of the legal values it is rounded to the nearest power of 2. If a number smaller than 31 or greater than 1024 is sent an error is produced.

The POINTS query returns the number of time buckets to be acquired.

Note Always query the WAVEFORM Subsystem points value to determine the actual number of time buckets acquired (:WAVEFORM:POINTS?).

Command Syntax

```
:ACQuire:POINts <points_arg>
```

Where:

```
<points_arg>::= 32 to 1024 (see above for legal values)
```

Example

```
OUTPUT 707;":ACQ:POINTS 512"
```

Query Syntax

```
:ACQuire:POINts?
```

Returned Format

```
<points_arg><NL>
```

Where:

```
<points_arg> ::= 32 - 1024 (see above for legal values)
```

Example

```
DIM Pnts$[50]
OUTPUT 707;":ACQUIRE:POINTS?"
ENTER 707;Pnts$
PRINT Pnts$
```

TYPE

The :ACQUIRE:TYPE command selects the type of acquisition that is to take place when a :DIGITIZE root level command is executed. There are three acquisition types: NORMAL, AVERAGE, and ENVELOPE.

The :ACQUIRE:TYPE query returns the current acquisition type.

Command Syntax

```
:ACQuire:TYPE {NORMal|AVERage|ENVelope}
```

Example

```
OUTPUT 707;":ACQUIRE:TYPE ENVELOPE"
```

Query Syntax

```
:ACQuire:TYPE?
```

Returned Format

```
<type><NL>
```

Where:

```
<type> ::= {NORMal|AVERage|ENVelope}
```

Example

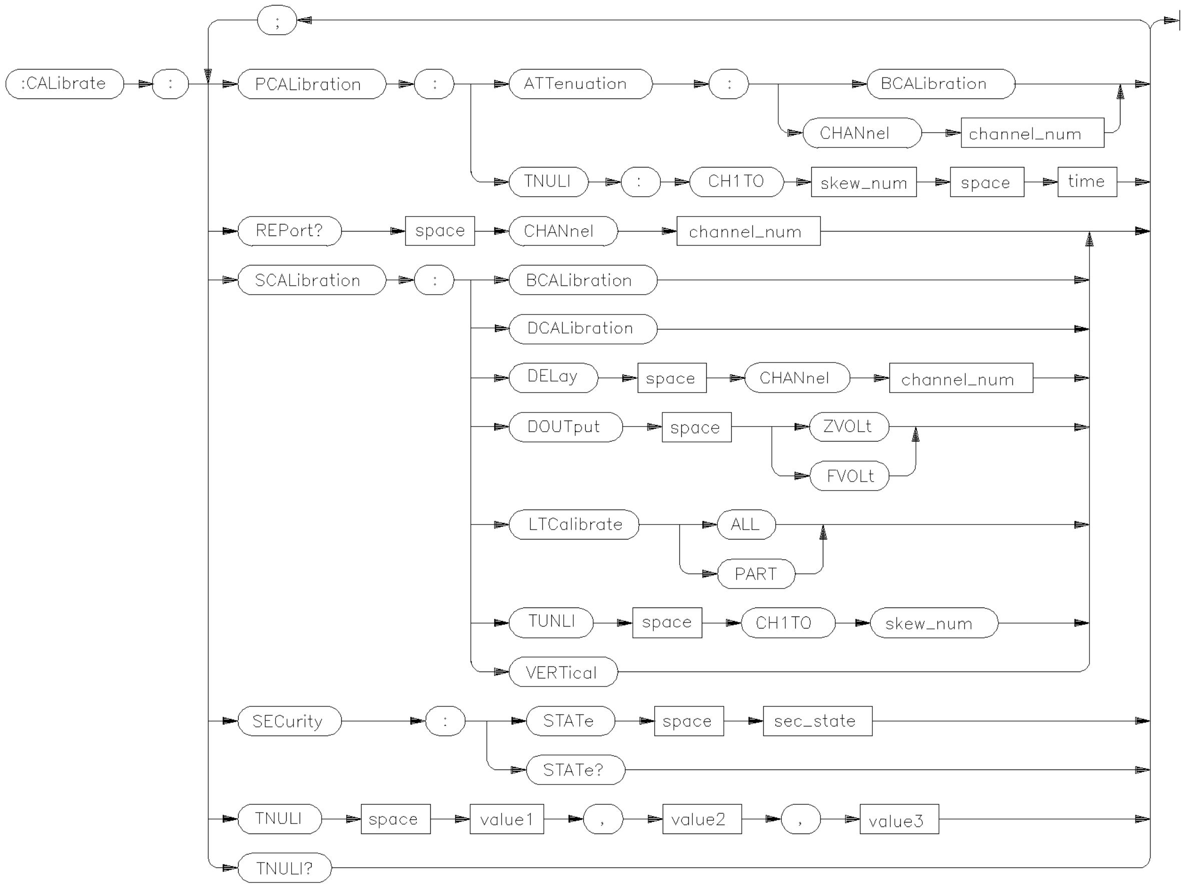
```
DIM Tpe$[50]
OUTPUT 707;":ACQUIRE:TYPE?"
ENTER 707;Tpe$
PRINT Tpe$
```

9

Calibrate Subsystem

The CALIBRATE subsystem contains commands to perform probe/self calibration, and set channel-to-channel time nulls. CALIBRATION may be used instead of CALIBRATE.

Note After running the SCALIBRATION command you must perform an AUTOSCALE or *RST command to return to normal operation.



channel_num	=	1 through 4
time	=	−50 nS to 70 nS
skew_num	=	2 through 4
sec_state	=	{{ON | 1} | {OFF | 0}}
value1	=	channel 1 to channel 2 skew
value2	=	channel 1 to channel 3 skew
value3	=	channel 1 to channel 4 skew

a70703a02

Figure 9-1. CALIBRATE Subsystem Commands Syntax Diagram

Calibration Memory Protection

The Calibration Memory Protection is controlled by switches A and B situated in the bank of switches at the top of the module. To set these switches, the mainframe must first be powered down, and then the HP 70703A removed from the mainframe to gain access to the switches.

Switch A determines whether the Calibration Memory Protection is controlled by the switch settings or by programmable methods. With switch A set to the "0" position ("hard mode"), the Calibration Memory Protection state is determined by the position of switch B. When B is set to "0", the Calibration Memory is write-protected. When B is set to "1", the Calibration Memory is not write-protected.

Setting switch A to "1" ("soft mode") means that the Calibration Memory Protection can be controlled by the programmable method. The position of switch B determines the power-up state of the Calibration Memory Protection. With switch B set to "0", the power-up state is write-protected. When B is set to "1", the power-up state is not write-protected. The initial power-up state can be altered using the :CALIBRATE:SECURITY:STATE command, this is particularly useful in ATE systems when it is not practical to power down the system. Sending :CALIBRATE:SECURITY:STATE ON will set the Calibration Memory Protection to write-protected mode. Sending :CALIBRATE:SECURITY:STATE OFF will set the Calibration Memory Protection to not write-protected mode. Cycling the power will reset the Calibration Memory Protection to the conditions determined by the positions of switches A and B.

It is recommended that the Calibration Memory Protection switches (A/B) are set to the write-protected settings, (A = 0 or 1, B = 0). This will ensure that at power-up the Calibration Memory Protection mode will be set to write protected.

The Calibration procedure for the HP 70703A is described in the Installation and Verification Manual.

PCALibration:ATTenuation:BCALibration

The :CALIBRATE:PCALIBRATION:ATTENUATION:BCALIBRATION command performs an attenuation calibration on the channel specified by the CAL:PCAL:ATT:CHAN<N> command. The instrument calibrates channel gain at the point connected to the DC calibrator output (DC CAL OUT) connector (probe, cable, and so on). Probe attenuation is then calculated from the results, and a correction is automatically entered in the correct CHANnel<N>:PROBe setting.

Command Syntax

```
:CALibrate:PCALibration:ATTenuation:BCALibration
```

Example

```
OUTPUT 707;":CAL:PCAL:ATT:CHAN4"
PAUSE   ! To connect probe to DC CAL OUT from Input 4 connector.
OUTPUT 707;":CAL:PCAL:ATT:BCAL"
```

This example calibrates the channel gain on input 4. For the example a 10:1 attenuator probe is connected to the DC CAL OUT connector. The correction is automatically stored in CHAN4:PROB.

Note Channel gain is corrected using calculated probe attenuation values from 0.9:1 to 250:1. If the measured results cause the calculated attenuation factor to be out of this range, an error will be generated.

PCALibration:ATTenuation:CHANnel

The :CALIBRATE:PCALIBRATION:ATTENUATION:CHANNEL<N> command selects the channel that will be calibrated when the CAL:PCAL:ATT:BCAL command is executed.

Command Syntax

```
:CALibrate:PCALibration:ATTenuation:CHANnel<N>
```

Where:

```
<N> ::= 1,2,3 or 4
```

Example

```
OUTPUT 707;":CAL:PCAL:ATT:CHAN2"
```

The :CALIBRATE:PCALIBRATION:TNULL:CH1TO<N> command is used to set the timing of channels 2, 3 or 4 to correspond with channel 1. Use to eliminate any time discrepancies between channels and minimize channel to channel skew variations. Use to manually adjust any differences in cable length.

Command Syntax

```
:CALibrate:PCALibration:TNULl:CH1TO<N> <time>
```

Where:

```
<N> ::= 2, 3 or 4 <time> ::= –50nS to 70nS
```

Example

```
OUTPUT 707;":CAL:PCAL:TNUL:CH1TO4 25E-9"
```

REPort

The :CALIBRATE:REPORT query returns the instrument's current calibration status. Each channel's status is queried separately. The data is sent to the output buffer.

Query Syntax

```
:CALibrate:REPort? <channel>
```

Where:

```
<channel> ::= {CHANnel{1|2|3|4}}
```

Returned Format

```
<data><NL>
```

Where:

```
<data> ::= {CHANnel1 A/D {P|F|D|C}, Gain {P|F|D|C},
           Offset {P|F|D|C}, Hysteresis {P|F|D|C}, Trigger {P|F|D|C},
           Delay {P|F|D|C}, Logic Trigger {P|F|D|C}|CHANnel{2|3|4}
           A/D {P|F|D|C}, Gain{P|F|D|C}, Offset {P|F|D|C},
           Hysteresis {P|F|D|C}, Trigger {P|F|D|C}, Delay {P|F|D|C},
           TimeNull {P|F|D|C}}
```

Note "P" = Passed, "F" = Failed, "D" = Defaulted, "C" = Corrupted. P,F,D or C prefixed by a "*", indicates a new ROM revision without a recalibration.

Example

```
DIM Data$[18000]
OUTPUT 707;":CAL:REPort? CHANnel2"
ENTER 707;Data$
PRINT Data$
```

SCALibration:BCALibration

The :CALIBRATE:SCALIBRATION:BCALIBRATION command is used to begin a self-calibration routine. The routine that is performed is dependent on the SCALIBRATION command configured prior to executing the BCALIBRATION command.

Command Syntax

```
:CALibrate:SCALibration:BCALibration
```

Example

```
OUTPUT 707;":CAL:SCAL:LTC"
OUTPUT 707;":CAL:SCAL:BCAL"
```

In this example, the logic trigger calibration has been configured with the CAL:SCAL:LTC command and the logic trigger calibration has been started by the CAL:SCAL:BCAL command.

Note The Calibration Protection circuitry must be set to the NOT WRITE PROTECTED mode prior to performing a SCALIBRATION routine. If the BCALIBRATION command is executed without first defining the SCALIBRATION routine to be performed, a bit will be set in the SUMMARY:QUESTIONABLE register (bit 8 - cal not active).

SCALibration:DCALibration

The :CALIBRATE:SCALIBRATION:DCALIBRATION command is used to load “default” calibration data. Default calibration data is set at the factory and is dependent on the ROM revision currently installed. This command should only be used by service personnel. Refer to the Service Manual for procedures to perform this calibration.

Command Syntax

```
:CALibrate:SCALibration:DCALibration
```

Example

```
OUTPUT 707;":CAL:SCAL:DCAL"
OUTPUT 707;":CAL:SCAL:BCAL"
```

This example overwrites all existing calibration data with default data.

Note The Calibration Protection circuitry must be set to the NOT WRITE PROTECTED mode prior to performing a default calibration routine.

SCALibration:DELay

The :CALIBRATE:SCALIBRATION:DELAY command performs a delay calibration on all four inputs, one at a time. Each input must be connected to the AC calibrator output (PROBE COMP AC CAL OUT) connector prior to executing the calibration routine for that channel. The results are stored and used by the instrument to maintain measurement accuracy.

Command Syntax

```
:CALibrate:SCALibration:DELay <channel>
```

Where:

```
<channel> ::= {CHANnel{1|2|3|4}}
```

Example

```
OUTPUT 707;":CAL:SCAL:DEL CHAN4"
OUTPUT 707;":CAL:SCAL:BCAL"
```

In this example, the instrument has been configured to perform a delay calibration on channel 4 with the CAL:SCAL:DEL command and the delay calibration has been started by the CAL:SCAL:BCAL command.

Note The Calibration Protection circuitry must be set to the NOT WRITE PROTECTED mode prior to performing a calibration routine.

SCALibration:DOUTput

The :CALIBRATE:SCALIBRATION:DOUTPUT command is used to set the output level of the DC calibrator output (DC CAL OUT) connector to 0 volts (ZVOLt) or 5 volts (FVOLt).

Command Syntax

```
:CALibrate:SCALibration:DOUTput {ZVOLt|FVOLt}
```

Example

```
OUTPUT 707;":CAL:SCAL:DOUT ZVOL"
```

Note The default condition after a *RST command is ZVOLt (0 volts).

SCALibration:LTCalibrate

The :CALIBRATE:SCALIBRATION:LTCALIBRATE command performs a logic trigger calibration. Input 1 must be connected to the AC calibrator output (PROBE COMP AC CAL OUT) connector prior to executing the calibration routine. The results are stored and used by the instrument to maintain measurement accuracy.

Executing :CAL:SCAL:LTC PART assumes the logic trigger oscillator is correctly adjusted, and only recalculates the internal constants. Executing :CAL:SCAL:LTC ALL requires the logic trigger oscillator to be adjusted.

Executing :CAL:SCAL:LTC without any argument is the same as executing the command with the ALL parameter.

This function is a service only function. Refer to the Service Manual for more information.

Command Syntax

```
:CALibrate:SCALibration:LTCalibrate {ALL|PART}
```

Example

```
OUTPUT 707;":CAL:SCAL:LTC"
OUTPUT 707;":CAL:SCAL:BCAL"
```

Note Prior to executing the logic trigger calibration routine, the calibration results must be reviewed using the CAL:REP? query. All four channel calibration results must indicate "P" before the logic trigger calibration can be executed. The Calibration Protection circuitry must be set to the NOT WRITE PROTECTED mode prior to performing a default calibration routine.

SCALibration:TNULl

The :CALIBRATE:SCALIBRATION:TNULL command performs a time null calibration on one set of channels at a time. The results are stored and used by the instrument to maintain measurement accuracy.

Command Syntax

```
:CALibrate:SCALibration:TNULl <channel skew>
```

Where:

```
<channel skew> ::= {CH1TO2|CH1TO3|CH1TO4}
```

Example

```
OUTPUT 707;":CAL:SCAL:TNUL CH1TO3"
OUTPUT 707;":CAL:SCAL:BCAL"
```

Note The Calibration Protection circuitry must be set to the NOT WRITE PROTECTED mode prior to performing a default calibration routine.

SCALibration:VERTical

The :CALIBRATE:SCALIBRATION:VERTICAL command performs a vertical calibration on all four inputs simultaneously. All inputs must be connected to the DC calibrator output (DC CAL OUT) connector prior to executing the calibration routine. The results are stored and used by the instrument to maintain measurement accuracy.

Command Syntax

```
:CALibrate:SCALibration:VERTical
```

Example

```
OUTPUT 707;":CAL:SCAL:VERT"
OUTPUT 707;":CAL:SCAL:BCAL"
```

Note The Calibration Protection circuitry must be set to the NOT WRITE PROTECTED mode prior to performing a default calibration routine.

SECurity:STATe

The :CALIBRATE:SECURITY:STATE is used to switch the Calibration Memory Protection mode between write-protected and not write-protected.

The SECURITY:STATE query returns the current state of Calibration Memory Protection.

Command Syntax

```
:CALibrate:SECurity:STATe {{ON|1}|{OFF|0}}
```

Example

```
OUTPUT 707;":CAL:SEC:STAT ON"
```

Query Syntax

```
:CALibrate:SECurity:STATe?
```

Returned Format

```
<sec_state>
```

Where:

```
<sec_state> ::= 0 or 1
```

Note The Calibration Protection state selected will remain valid until a new selection is made or until the power is cycled. Cycling the power will return the Calibration Protection mode to the power up state determined by the positions of switches A and B.

When the module is configured for "hard" write-protected mode (switch A set to "0") sending the :CALIBRATE: SECURITY:STATE command will always result in an error being returned. This mode cannot be changed programmatically, regardless of the state settings (switch B position).

TNULl

The :CALIBRATE:TNULL command sends the time null (channel-to-channel skew) values into the HP 70703A. The time null values should have been obtained from the instrument during a previous setup. The TNULL query tells the instrument to output the time null values to the controller.

Command Syntax

```
:CALibrate:TNULl <null_value1>,<null_value2>,<null_value3>
```

Where:

```
<null_value1> ::= channel 1 to channel 2 skew
<null_value2> ::= channel 1 to channel 3 skew
<null_value3> ::= channel 1 to channel 4 skew
```

Example

```
OUTPUT 707;":CAL:TNUL <null_value1>,<null_value2>,<null_value3>
```

Query Syntax

```
:CALibrate:TNULl?
```

Returned Format

```
<null_value1>,<null_value2>,<null_value3><NL>
```

Where:

```
<null_value1> ::= channel 1 to channel 2 skew (exponential - NR3 format)
<null_value2> ::= channel 1 to channel 3 skew (exponential - NR3 format)
<null_value3> ::= channel 1 to channel 4 skew (exponential - NR3 format)
```

Example

```
DIM Nll$[50]
OUTPUT 707;":CAL:TNUL?
ENTER 707;Nll$
PRINT Nll$
```

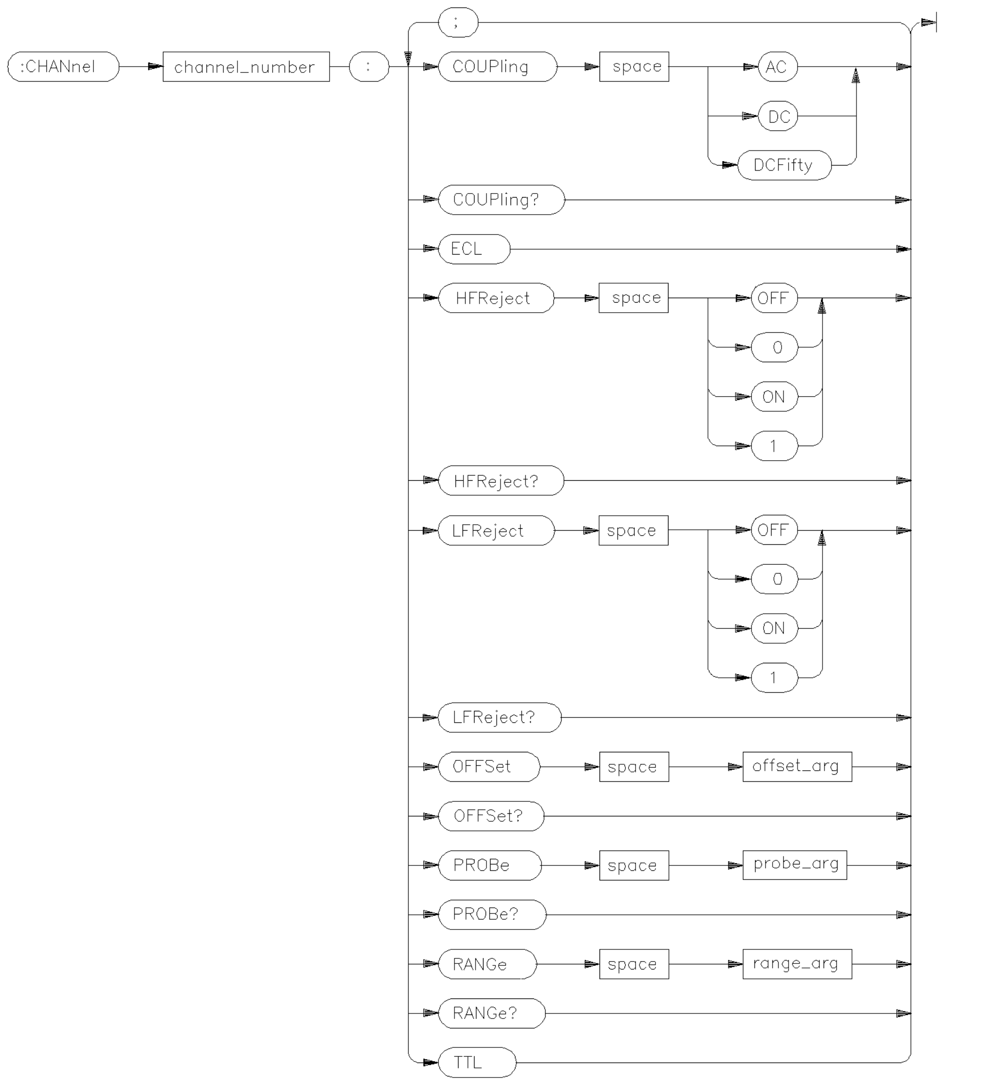
10

Channel Subsystem

The CHANNEL subsystem commands are used to select a specific channel's vertical or Y-axis control. Channels 1, 2, 3, and 4 are independently programmable for all offset, probe, coupling, and range functions.

The CHANNEL commands can be sent with a channel number specified or not specified. If a channel number is specified in the command, then the specified channel is affected. However, if the channel number is not specified, then channel 1 is affected.

See VIEW and BLANK for information on channel presentation.



channel_number	=	1 through 4
offset_arg	=	real number (defines voltage at center of display range)
probe_arg	=	real number, 0.9 to 1000.0 (specifies probe attenuation with respect to 1)
range_arg	=	real number (specifies size of acquistion window in volts)

a70703a29

Figure 10-1. CHANNEL Subsystem Commands Syntax Diagram

COUPling

The :CHANNEL<N>:COUPLING command selects the input coupling for the specified channel. The coupling for each channel can be set to AC, DC, or DCFIFTY. DCFIFTY places an internal fifty ohm load on the input.

The COUPLING query returns the current coupling for the specified channel.

Command Syntax

```
:CHANnel<N>:COUPling {AC|DC|DCFifty}
```

Where:

```
<N> ::= 1, 2, 3, or 4
```

Example

```
OUTPUT 707;":CHAN2:COUP DC"
```

Query Syntax

```
:CHANnel<N>:COUPling?
```

Where:

```
<N> ::= 1, 2, 3, or 4
```

Returned Format

```
{AC|DC|DCFifty}<NL>
```

Example

```
DIM Ch$[50]
OUTPUT 707;":CHAN2:COUPLING?"
ENTER 707;Ch$
PRINT Ch$
```

ECL

The :CHANNEL<N>:ECL command sets the vertical range, offset, channel coupling, and trigger level of the selected channel for optimum viewing of ECL signals. The offset and trigger level are set to − 1.3 volts and the range is set to 1.6 volts full-scale. Channel coupling is set to DC.

There is no query form of this command.

Command Syntax

```
:CHANnel<N>:ECL

Where:

<N> ::= 1, 2, 3, or 4
```

Example

```
OUTPUT 707;":CHAN1:ECL"
```

HFReject

The :CHANNEL<N>:HFREJECT command controls an internal low-pass filter. When ON the bandwidth of the specified channel is limited to approximately 20 MHz. The bandwidth limit filter may be used when either AC, DC, or DCFIFTY coupling is used.

The HFREJECT query returns the current setting.

Command Syntax

```
:CHANnel<N>:HFReject {{ON|1}|{OFF|0}}
```

Where:

```
<N> ::= 1, 2, 3, or 4
```

Example

```
OUTPUT 707;":CHANNEL2:HFR ON"
```

Query Syntax

```
:CHANnel<N>:HFReject?
```

Where:

```
<N> ::= 1, 2, 3, or 4
```

Returned Format

```
{1|0}<NL>
```

Example

```
DIM Hf$[50]
OUTPUT 707;":CHAN:HFR?"
ENTER 707;Hf$
PRINT Hf$
```

LFReject

The :CHANNEL<N>:LFREJECT command controls an internal high-pass filter. When ON the bandwidth of the specified channel is limited to approximately 400 Hz. The bandwidth limit filter may be used only when AC coupling is used.

The LFREJECT query returns the current setting.

Command Syntax

```
:CHANnel<N>:LFReject {{ON|1}|{OFF|0}}
```

Where:

```
<N> ::= 1, 2, 3, or 4
```

Example

```
OUTPUT 707;":CHANNEL2:LFR ON"
```

Query Syntax

```
:CHANnel<N>:LFReject?
```

Where:

```
<N> ::= 1, 2, 3, or 4
```

Returned Format

```
{1|0}<NL>
```

Example

```
DIM Lf$[50]
OUTPUT 707;":CHAN:LFR?"
ENTER 707;Lf$
PRINT Lf$
```

OFFSet

The :CHANNEL<N>:OFFSET command sets the voltage that is represented at the center of the current range for the selected channel.

The range of legal values varies with the value set with the RANGE command. If you set the offset to a value outside the legal range, it will automatically be set to the nearest legal value.

The OFFSET query returns the current offset value for the selected channel.

Command Syntax

```
:CHANnel<N>:OFFSet <value>
```

Where:

```
<N> ::= 1, 2, 3, or 4 <value> ::= offset value
```

Example

```
OUTPUT 707;":CHAN1:OFFS 200M;:CHAN2:OFFSET 20E-3"
```

Query Syntax

```
:CHANnel<N>:OFFSet?
```

Where:

```
<N> ::= 1,2,3, or 4
```

Returned Format

```
<value><NL>
```

Where:

```
<value> ::=  offset value in volts (exponential - NR3 format)
```

Example

```
OUTPUT 707;":CHANNEL2:OFFSET?"
ENTER 707;Offset
PRINT Offset
```

PROBe

The :CHANNEL<N>:PROBE command specifies the probe attenuation factor for the selected channel. The range of the probe attenuation factor is from 0.9 to 1000.0. This command does not change the actual input sensitivity of the HP 70703A. It changes the reference constants for scaling the display factors and for automatic measurements, trigger levels, and so on.

The PROBE query returns the current probe attenuation factor for the selected channel.

Command Syntax

```
:CHANnel<N>:PROBe <atten>
```

Where:

```
<N> ::= 1, 2, 3, or 4 <atten> ::= 0.9 to 1000
```

Example

```
OUTPUT 707;":CHANNEL2:PROBE 10"
```

Query Syntax

```
:CHANnel<N>:PROBe?
```

Where:

```
<N> ::= 1, 2, 3, or 4
```

Returned Format

```
<atten><NL>
```

Where:

```
<atten> ::= 0.9 to 1000 (exponential - NR3 format)
```

Example

```
DIM Prb$[50]
OUTPUT 707;":CHANNEL1:PROBE?"
ENTER 707;Prb$
PRINT Prb$
```

RANGe

The :CHANNEL<N>:RANGE command defines the full-scale vertical axis of the selected channel. The RANGE can be set to any value from 8 mV to 40 V, when using 1:1probe attenuation. If the probe attenuation is changed, the range value is multiplied by the probe attenuation factor.

The RANGE query returns the current range setting for the specified channel.

Command Syntax

```
:CHANnel<N>:RANGe <range>
```

Where:

```
<N> ::= 1, 2, 3, or 4
<range> ::= range value
```

Examples

```
:OUTPUT 707;"CHANNEL1:RANGE .60"

:OUTPUT 707;"CHANNEL2:RANGE 1.2 V"
```

Query Syntax

```
:CHANnel<N>:RANGe?
```

Where:

```
<N>::= 1, 2, 3, or 4
```

Returned Format

```
<range><NL>
```

Where:

```
<range> ::= range value (exponential - NR3 format)
```

Example

```
DIM Rng$[50]
OUTPUT 707;":CHAN2:RANGE?"
ENTER 707;Rng$
PRINT Rng$
```

TTL

The :CHANNEL<N>:TTL command sets the vertical range, offset, channel coupling, and trigger level of the selected channel for optimum viewing of TTL signals. The offset is set to 2.5 volts. The trigger level is set to 1.4 volts and the range is set to 8.0 volts full-scale. Channel coupling is set to DC.

There is no query form of this command.

Command Syntax

```
:CHANnel<N>:TTL
```

Example

```
OUTPUT 707;":CHAN1:TTL"
```

Where:

```
<N> ::= 1, 2, 3, or 4
```

11

Display Subsystem

The DISPLAY subsystem is used to control the display of data on the screen.

Note The command that changes the display mode is :ACQUIRE:TYPE. The command that controls the number of averages is :ACQUIRE:COUNT.

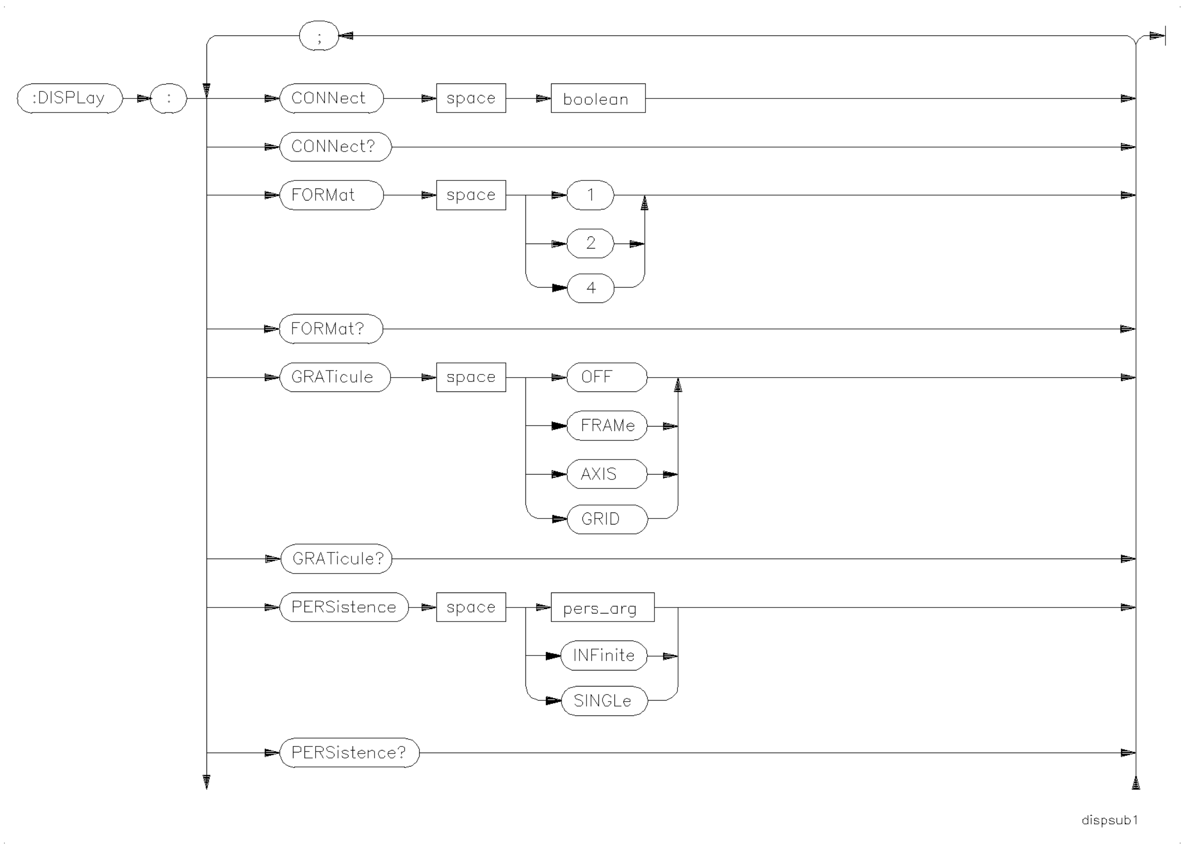
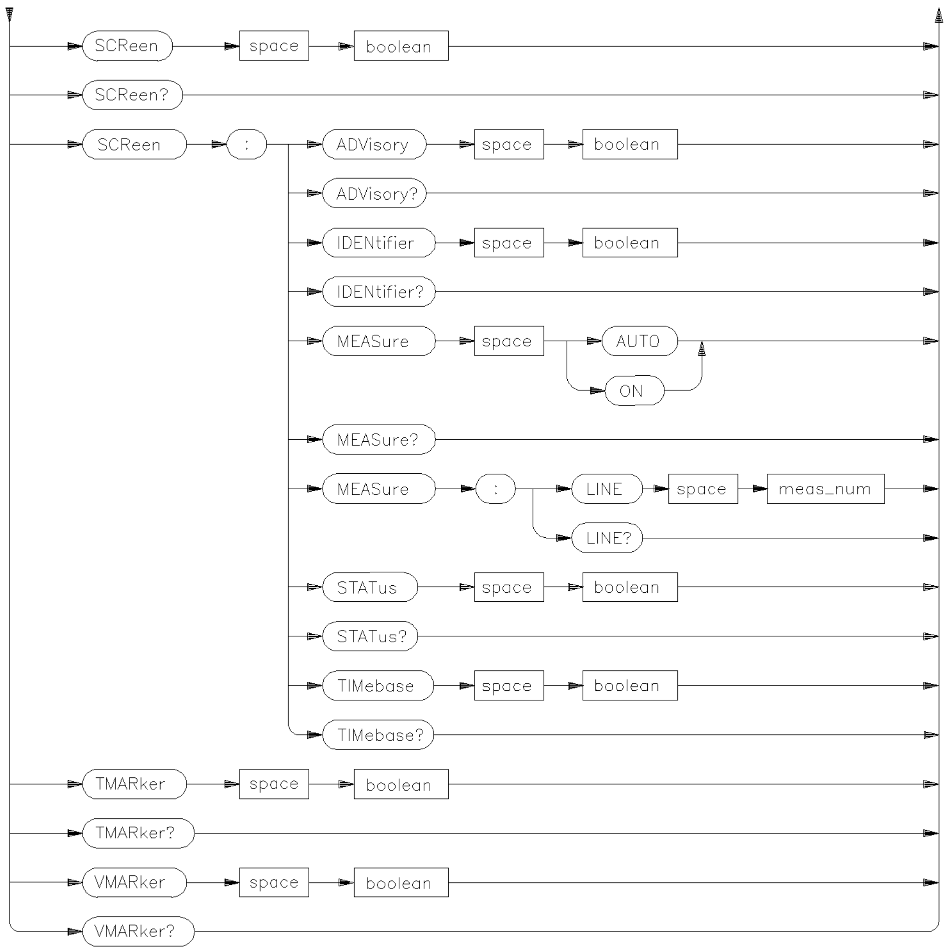


Figure 11-1. DISPLAY Subsystem Commands Syntax Diagram



boolean = {{ON | 1} | {OFF | 0}}

meas_num = maximum number of displayed measurements, 1..8

pers_arg = real number, <0.15 and >10.00

dispsub2

Figure 11-2. DISPLAY Subsystem Commands Syntax Diagram (continued)

CONNect

The :DISPLAY:CONNECT command turns the connect-the-dots function on and off.

The CONNECT query returns the current setting of the connect-the-dots function. The returned status is indicated by using a 1 for on and a 0 for off.

Command Systax

```
:DISPlay:CONNect {{ON|1}|{OFF|0}}
```

Example

```
OUTPUT 707;":DISPLAY:CONNECT ON"
```

Query Syntax

```
:DISPlay:CONNect?
```

Returned Format

```
{1|0}<NL>
```

Example

```
DIM Cnn$[50]
OUTPUT 707;":DISP:CONNECT?"
ENTER 707;Cnn$
PRINT Cnn$
```

FORMat

The :DISPLAY:FORMAT command sets the number of display areas on the CRT. Format 1 provides one display area containing all four channels. Format 2 provides two display areas with channels 1 and 2 in the upper area and 3 and 4 in the lower area. Format 4 provides four display areas with one channel per area.

The FORMAT query returns the current display format.

Command Syntax

```
:DISPlay:FORMat {1|2|4}
```

Example

```
OUTPUT 707;":DISP:FORMAT 1"
```

Query Syntax

```
:DISPlay:FORMat?
```

Returned Format

```
{1|2|4}<NL>
```

Example

```
DIM Frmt$[30]
OUTPUT 707;":DISPLAY:FORMAT?"
ENTER 707;Frmt$
PRINT Frmt$
```

GRATicule

The :DISPLAY:GRATICULE command selects the type of graticule that is displayed. The GRATICLUE query returns the type of graticule displayed.

Command Syntax

```
:DISPlay:GRATiclue {OFF|GRID|AXES|FRAMe}
```

Example

```
OUTPUT 707;":DISPLAY:GRATICULE AXES"
```

Query Syntax

```
:DISPlay:GRATiclue?
```

Returned Format

```
<type><NL>
```

Where:

```
<type>::= {OFF|GRID|AXES|FRAMe}
```

Example

```
DIM Grt$[30]
OUTPUT 707;":DISPLAY:GRATICULE?"
ENTER 707;Grt$
PRINT Grt$
```

PERSistence

The :DISPLAY:PERSISTENCE command sets the display persistence for the current display.

The parameters for this command are the keywords INFINITE, SINGLE, or a real number. A real number less than 0.15 sets the persistence to SINGLE, and a real number greater than 10.00 sets the persistence to INFINITE. All other real numbers generate the error message –222, `"Data out of range"`.

When SINGLE is selected, the contents of the current display are updated to reflect the last data point hit in a time bucket. When INFINITE is selected, all the contents of the current display are retained and current waveform data is added.

The PERSISTENCE query returns the current persistence value. When minimum is displayed, the value returned is 0. When infinite is displayed, the value returned is 11. Both values are returned in NR3 exponential format.

Command Syntax

```
:DISPlay:PERSistence {INFinite|SINGle|0.1|11.0}
```

Example

```
OUTPUT 707;":DISPLAY:PERSISTENCE INFINITE"
```

Query Syntax

```
:DISPlay:PERSistence?
```

Returned Format

```
<value><NL>
```

Where:

```
<value> ::= {0|11} (NR3 exponential format)
```

Example

```
DIM Prs$[50]
OUTPUT 707;":DISPLAY:PERSISTENCE?"
ENTER 707;Prs$
PRINT Prs$
```

SCReen

The :DISPLAY:SCREEN command turns the displayed screen on and off. The screen can be turned on again with the ON parameter. The SCREEN query returns the current setting of this function. The returned status is indicated by using a 1 for on and a 0 for off.

Command Syntax

```
:DISPlay:SCReen {{ON|1}|{OFF|0}}
```

Example

```
OUTPUT 707;":DISPLAY:SCREEN ON"
```

Query Syntax

```
:DISPlay:SCReen?
```

Returned Format

```
{1|0}<NL>
```

Example

```
DIM Srn$[50]
OUTPUT 707;":DISP:SCREEN?"
ENTER 707; Srn$
PRINT Srn$
```

SCReen:ADVisory

The :DISPLAY:SCREEN:ADVISORY command turns the advisory message on the screen on and off. When the area is off, advisories are not displayed.

Command Syntax

```
:DISPlay:SCReen:ADVisory {{ON|1}|{OFF|0}}
```

Example

```
OUTPUT 707;":DISP:SCR:ADV ON"
```

Query Syntax

```
:DISPlay:SCReen:ADVisory?
```

Returned Format

```
{1|0}<NL>
```

Where:

```
1::= ON
2::= OFF
```

Example

```
OUTPUT 707;":DISP:SCR:ADV?"
ENTER 707;State
PRINT State
```

SCReen:IDENtifier

The :DISPLAY:SCREEN:IDENTIFIER command turns the identifier message area on the screen on and off. When the area is off, the module identifier is not displayed.

Command Syntax

```
:DISPlay:SCReen:IDENtifier {{ON|1}|{OFF|0}}
```

Example

```
OUTPUT 707;":DISP:SCR:IDEN ON"
```

Query Syntax

```
:DISPlay:SCReen:IDENtifier?
```

Returned Format

```
{1|0}<NL>
```

Where:

```
1::= ON
0::= OFF
```

Example

```
OUTPUT 707;":DISP:SCR:IDEN?"
ENTER 707;State
PRINT State
```

SCReen:MEASure

The :DISPLAY:SCREEN:MEASURE command affects the area below the traces where the measurment results are displayed. When in AUTO mode the measurement area is only allocated (for example, only reduces the area available to trace display) when measurements are being displayed. When in ON mode the measurement area is permanently allocated.

Command Syntax

```
:DISPlay:SCReen:MEASure {AUTO|ON}
```

Example

```
OUTPUT 707;"DISP:SCR:MEAS AUTO"
```

Query Syntax

```
:DISPlay:SCReen:MEASure?
```

Returned Format

```
{AUTO|ON}<NL>
```

Example

```
DIM State$[10]
OUTPUT 707;":DISP:SCR:MEAS?"
ENTER 707;State$
PRINT State$
```

SCReen:MEASure:LINE

The :DISPLAY:SCREEN:MEASURE:LINE command affects the area below the traces where the measurement results are displayed. The parameter denotes the maximum number of measurements which will be displayed by the instrument in the measurment area. This indirectly affects the size of the measurement area allocated. See also :DISPLAY:SCREEN:MEASURE command.

Command Syntax

```
:DISPlay:SCReen:MEASure:LINE <max_meas>
```

Where:

```
<max_meas> ::=maximum number of measurements, 1 to 8
```

Example

```
OUTPUT 707;":DISP:SCR:MEAS:LINE 8"
```

Query Syntax

```
:DISPlay:SCReen:MEASure:LINE?
```

Returned Format

```
<max_meas><NL>
```

Example

```
OUTPUT 707;":DISP:SCR:MEAS:LINE?"
ENTER 707;Max_meas
PRINT Max_meas
```

SCReen:STATus

The :DISPLAY:SCREEN:STATUS command turns the status message area on the display screen on and off. When the area is off, acqusition status in not displayed.

Command Syntax

```
DISPlay:SCReen:STATus {{ON|1}|{OFF|0}}
```

Example

```
OUTPUT 707;":DISP:SCR:STAT ON"
```

Query Syntax

```
:DISPlay:SCReen:STATus?
```

Returned Format

```
{1|0}<NL>
```

Where:

```
1::= ON
0::= OFF
```

Example

```
OUTPUT 707;":DISP:SCR:STAT?"
ENTER 707;State
PRINT 707 State
```

SCReen:TIMebase

The :DISPLAY:SCREEN:TIMEBASE command turns the timebase scaling text area on the display screen on and off. When the area is off, the current timebase range scale is not displayed.

Command Syntax

```
:DISPlay:SCReen:TIMebase {{ON|1}|{OFF|0}}
```

Example

```
OUTPUT 707;":DISP:SCR:TIM ON"
```

Query Syntax

```
:DISPlay:SCReen:TIMebase?
```

Returned Format

```
{1|0}<NL>
```

Where:

```
1::= ON
0::= OFF
```

Example

```
OUTPUT 707;":DISP:SCR:TIM?"
ENTER 707;State
PRINT 707 State
```

TMARker

The :DISPLAY:TMARKER command turns the time markers on and off. The TMARKER query returns the state of the time markers.

Note It is a recommended practice to turn the time markers on before attempting to set them using a :MEASURE:TSTART or :MEASURE:TSTOP command.

Command Syntax

```
:DISPlay:TMARker {{ON|1}|{OFF|0}}
```

Example

```
OUTPUT 707;":DISP:TMAR OFF"
```

Query Syntax

```
:DISPlay:TMARker?
```

Returned Format

```
{1|0}<NL>
```

Example

```
DIM Tmr$[30]
OUTPUT 707;":DISP:TMARKER?"
ENTER 707;Tmr$
PRINT Tmr$
```

VMARker

The :DISPLAY:VMARKER command turns the voltage markers on and off. The VMARKER query returns the state of the voltage markers.

Note It is a recommended practice to turn the voltage markers on before attempting to set them using a :MEASURE:VSTART or :MEASURE:VSTOP command.

Command Syntax

```
:DISPlay:VMARker {{ON|1}|{OFF|0}}
```

Example

```
OUTPUT 707;":DISP:VMARKER ON"
```

Query Syntax

```
:DISPlay:VMARker?
```

Returned Format

```
{1|0}<NL>
```

Example

```
DIM Vmrk$[30]
OUTPUT 707;":DISP:VMARKER?"
ENTER 707;Vmrk$
PRINT Vmrk$
```

12

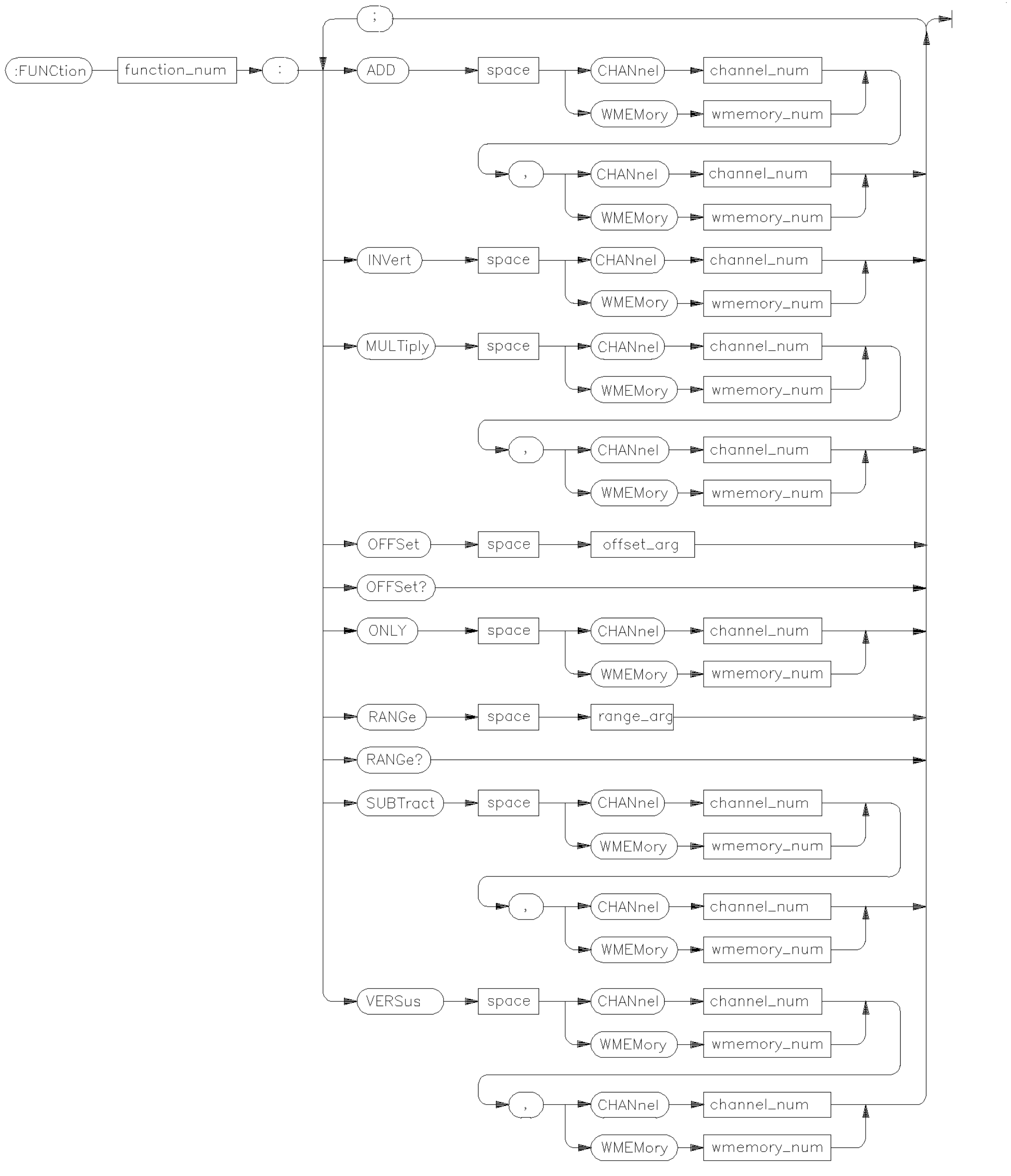
Function Subsystem

The FUNCTION subsystem defines six functions that use signals acquired on channels 1 to 4 and/or stored in waveform memories 1 to 4 as operands to create altered or duplicate waveforms. The operators are ADD, SUBTRACT, MULTIPLY, VERSUS, ONLY, and INVERT.

If a channel or memory that is not on is specified as an operand, then that channel is enabled.

Refer to figure 12-1 for a syntax diagram of the FUNCTION subsystem commands.

Channel 1 through 4 and Waveform Memories 1 through 4 are available for functions.



channel_num = integer, 1 through 4
function_num = 1 or 2
offset_arg = 0 to $\pm$ voltage full scale
range_arg = full screen voltage
wmemory_num = integer, 1 through 4

a70703a30

Figure 12-1. FUNCTION Subsystem Commands Syntax Diagram

ADD

The :FUNCTION <N>:ADD command algebraically sums the two defined operands.

Command Syntax

```
:FUNCtion<N>:ADD <operand>,<operand>
```

Where:

```
<N> ::= 1 or 2

<operand> ::= {CHANnel1|CHANnel2|CHANnel3|CHANnel4|
              WMEMory1|WMEMory2|WMEMory3|WMEMory4}
```

Example

```
OUTPUT 707;":FUNCTION2:ADD WMEMORY3,WMEMORY4"
```

INVert

The :FUNCTION<N>:INVERT command inverts the operand.

Command Syntax

```
:FUNCtion<N>:INVert <operand>
```

Where:

```
<N> ::= 1 or 2
<operand> ::= {CHANnel1|CHANnel2|CHANnel3|CHANnel4|
              WMEMory1|WMEMory2|WMEMory3|WMEMory4}
```

Example

```
OUTPUT 707;":FUNCTION2:INVERT WMEMORY3"
```

MULTiply

The :FUNCTION<N>:MULTIPLY command causes the instrument to algebraically multiply the two operands.

Command Syntax

```
:FUNCtion<N>:MULTiply <operand>,<operand>
```

Where:

```
<N> ::= 1 or 2
<operand> ::= {CHANnel1|CHANnel2|CHANnel3|CHANnel4|
              WMEMory1|WMEMory2|WMEMory3|WMEMory4}
```

Example

```
OUTPUT 707;":FUNCTION2:MULTIPLY CHANNEL1,CHANNEL2"
```

OFFSet

The :FUNCTION<N>:OFFSET command sets the voltage that is represented at the center of the current range for the selected function. The maximum range of acceptable offset values is ± the current FUNCTION<N>:RANGE setting.

The OFFSET query returns the current offset value for the selected function.

Command Syntax

```
:FUNCtion<N>:OFFSet <offset>
```

Where:

```
<N> ::= 1 or 2 <offset> ::= offset value (see above)
```

Example

```
OUTPUT 707;":FUNCTION1:OFFSET 650E-4"
```

Query Syntax

```
:FUNCtion<N>:OFFSet?
```

Where:

```
<N> ::= 1 or 2
```

Returned Format

```
<offset><NL>
```

Where:

```
<offset> ::=  offset value (see above) (exponential - NR3 format)
```

Example

```
OUTPUT 707;":FUNCTION2:OFFSET?"
ENTER 707;Off
PRINT Off
```

ONLY

The :FUNCTION<N>:ONLY command is used to make another copy of the operand. The ONLY command is useful for scaling channels and memories with the :FUNCTION<N>:RANGE and :FUNCtION<N>:OFFSET commands.

Command Syntax

```
:FUNCtion<N>:ONLY<operand>
```

Where:

```
<N> ::= 1 or 2
<operand> ::= {CHANnel1|CHANnel2|CHANnel3|CHANnel4|
               WMEMory1|WMEMory2|WMEMory3|WMEMory4}
```

Example

```
OUTPUT 707;":FUNCTION2:ONLY WMEMORY4"
```

RANGe

The :FUNCTION<N>:RANGE command defines the full-scale vertical axis of the selected function.

The RANGE query returns the current range setting for the specified function.

Command Syntax

```
:FUNCtion<N>:RANGe <range>
```

Where:

```
<N> ::= 1 or 2
<range> ::= voltage value
```

Example

```
OUTPUT 707;":FUNCTION2:RANGE 400 MV"
```

Query Syntax

```
:FUNCtion<N>:RANGe?
```

Where:

```
<N> ::= 1 or 2
```

Returned Format

```
<range><NL>
```

Where:

```
<range> ::= current range setting (exponential - NR3 format)
```

Example

```
OUTPUT 707;":FUNCTION2:RANGE?"
ENTER 707;Rng
PRINT Rng
```

SUBTract

The :FUNCTION<N>:SUBTRACT command algebraically subtracts operand 2 from operand 1.

Command Syntax

```
:FUNCtion<N>:SUBTract <operand>,<operand>
```

Where:

```
<N> ::= 1 or 2
<operand> ::= {CHANnel1|CHANnel2|CHANnel3|CHANnel4|
              WMEMory1|MEMory2|WMEMory3|WMEMory4}
```

Example

```
OUTPUT 707;":FUNCTION2:SUBTRACT WMEMORY3,WMEMORY2"
```

In this example, Waveform Memory 2 is algebraically subtracted from Waveform Memory 3.

VERSus

The :FUNCTION<N>:VERSUS command allows X vs Y displays with two operands. The first operand defines the Y axis and the second defines the X axis. The Y axis range and offset is initially equal to that of the first operand and can be adjusted with the FUNCTION<N>:RANGE and FUNCTION<N>:OFFSET commands.

Command Syntax

```
:FUNCtion<N>:VERSus<Y_operand>,<X_operand>
```

Where:

```
<N>::= 1 or 2

<Y_operand>::={CHAnel1|CHANnel2|CHANnel3|CHANnel4|WMEMory1|
              WMEMory2|WMEMory3|WMEMory4}

<X_operand>::={CHANnel1|CHANnel2|CHANnel3|CHANnel4|WMEMory1|
              WMEMory2|WMEMory3|WMEMory4}
```

Example

```
OUTPUT 707;":FUNCTION2:VERSUS CHAN1,CHAN2"
```

13

Measure Subsystem

The MEASURE subsystem is used to make parametric measurements on the specified source, and to return the measured voltage and time values. Measured results (up to eight) are retained in a measurement queue. Voltage, time, and event markers are automatically positioned during measurement, or can be manually set to specify voltages, times, or events.

Measurement Setup

To make a measurement, the portion of the waveform required for that measurement must be present on the active waveform. That is:

- For a period or frequency measurement, at least one complete cycle must be present.
- For a pulse width measurement, the entire pulse must be present.
- For a risetime measurement, the leading (positive-going) edge of the waveform must be present.
- For a falltime measurement, the trailing (negative-going) edge of the waveform must be present.

Note When WINDOW is ON, measurements are ONLY applied to the windowed portion of the waveform.

User-Defined Measurements

When user-defined measurements are made, the defined parameters must be set before actually sending the measurement command or query.

In user-defined measurements, the mid threshold is the mid point between the upper and lower threshold when the lower threshold value is less than the upper threshold value.

Measurement Error

If a measurement cannot be made, the value returned for that parameter is +9.99999E+37. This is an error value that is output when a measurement cannot be made.

Making Measurements

If more than one waveform, edge, or pulse is present, time measurements are made on the first (left-most) portion of the active waveform that can be used. When any of the defined measurements are requested, the oscilloscope first determines the top (100%) and base (0%) voltages of the waveform.

From this information, it can determine the other important voltage values (10% voltage, 90% voltage, and 50% voltage) for making the measurements. The 10% and 90% voltage values are used in the risetime and falltime measurements when standard measurements are selected. The 50% voltage value is used for measuring frequency, period, pulse width, and duty cycle with standard measurements selected.

The measurements can also be made using user-defined parameters instead of the standard measurement values.

When the commands PRESHOOT, or OVERSHOOT are used the, instrument is placed into the continuous measurement mode. When the query form of these measurements is used continuous measurement mode is turned off, the measurement is made one time and the measurement result is returned.

Voltage measurements are made using the entire active waveform. Therefore, if you want to make a measurement on a particular cycle, use :TIMEBASE:WINDOW to activate the expanded time base.

All voltage values are returned in volts. Returned voltage values are measured with zero volts as the reference. The value returned for the VDELTA? query is the difference between VMarker 1 and VMarker 2 in volts.

All time values are returned in seconds. Returned time values are measured with the trigger point (time 0) as the reference. The value returned for TDELTA? is the time difference between the stop and start markers.

Measurements are made on the active waveform(s) specified by the SOURCE command. The SOURCE command allows two sources to be specified. When two sources are specified, Vmarker 1 is assigned to the first specified source and VMarker 2 is assigned to the second specified source. VDELTA is the only measurement that uses two sources.

Most measurements can only be made on a single source. If one of these measurements is made with two sources specified, the measurement is made on the first source specified.

More information about measurement algorithms can be found in Appendix A.

If the horizontal scaling is questionable, bit 0 of the :SUMMARY:QUESYIONABLE:TIME register is set (refer to the SUMMARY subsystem). In this case, the value returned is the most accurate that can be made using the current scaling. You might be able to obtain a more accurate measurement by rescaling the horizontal to obtain more data points on the edge.

Refer to figure 13-1 for the syntax diagram of the MEASURE subsystem commands.

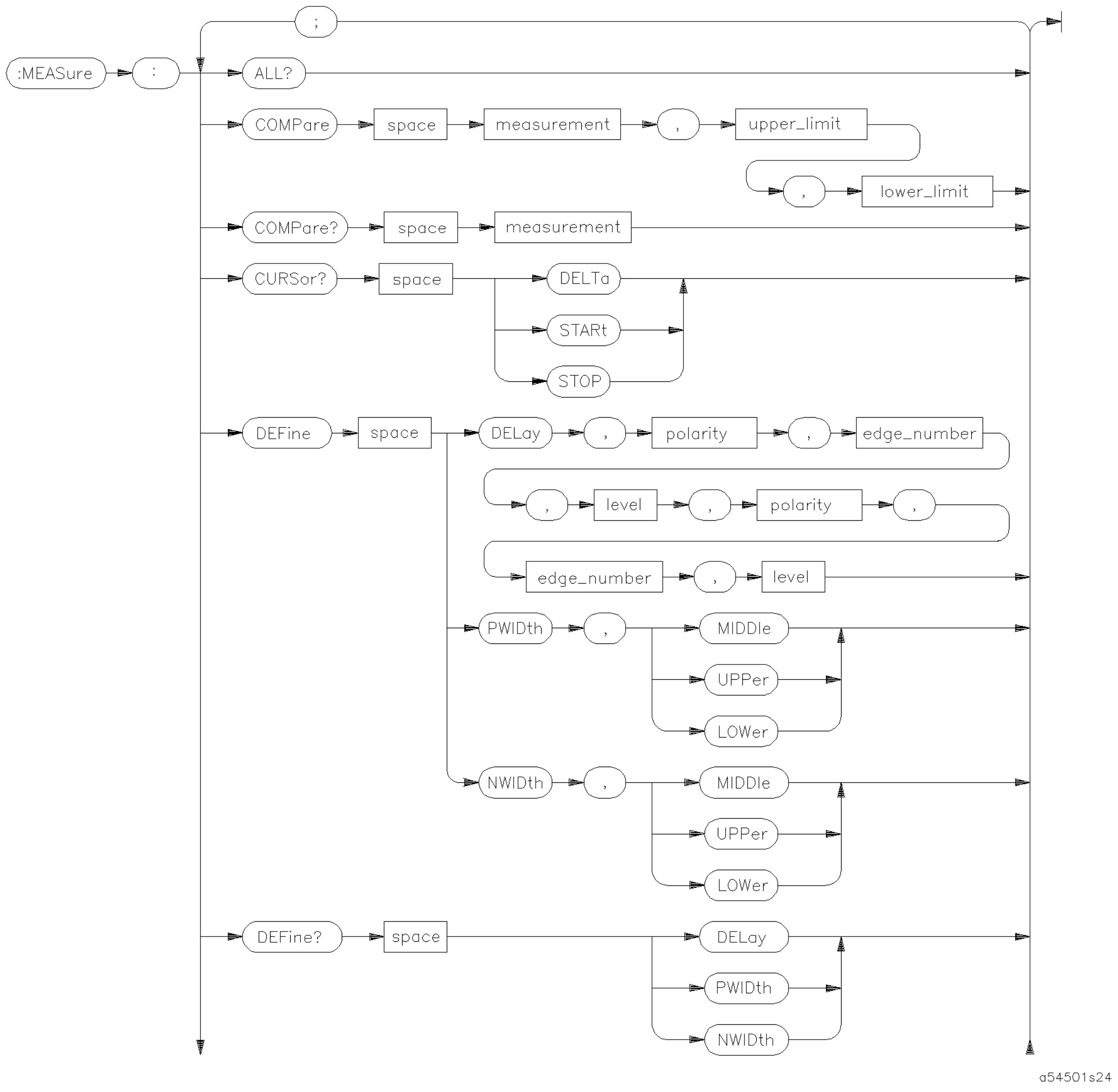


Figure 13-1. MEASURE Subsystem Commands Syntax Diagram

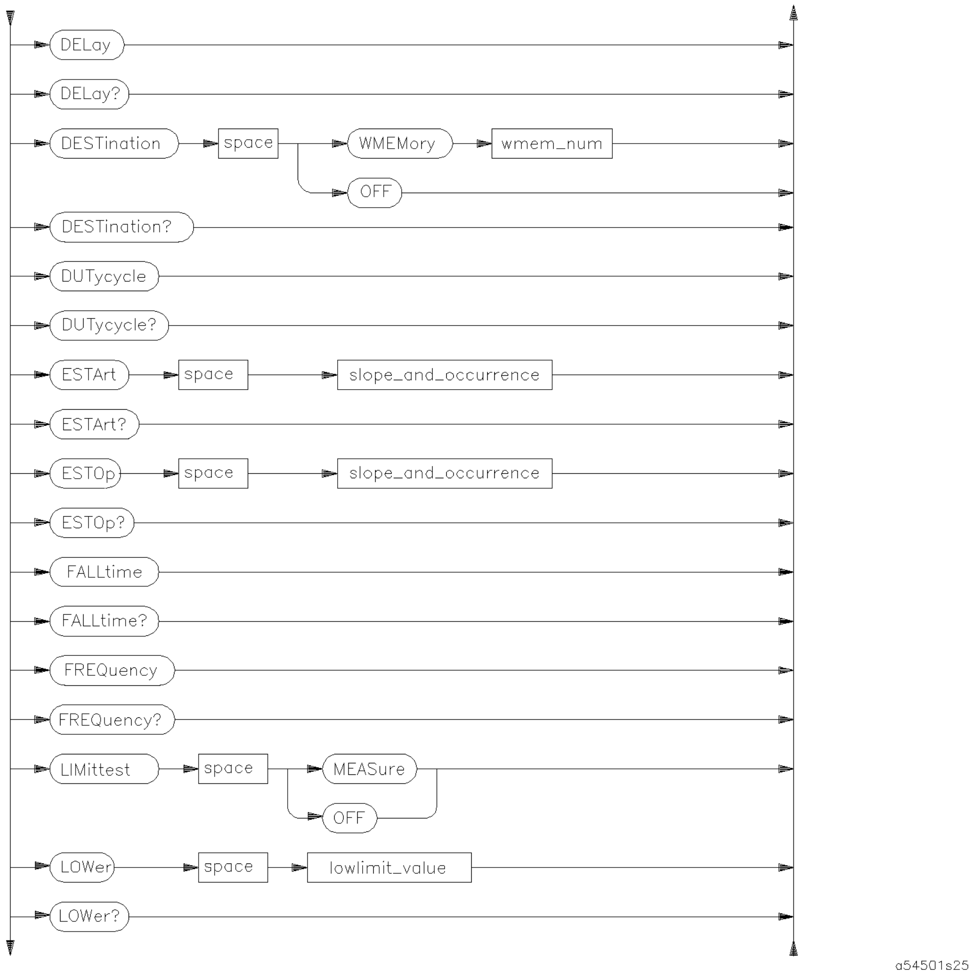


Figure 13-1. MEASURE Subsystem Commands Syntax Diagram (continued)

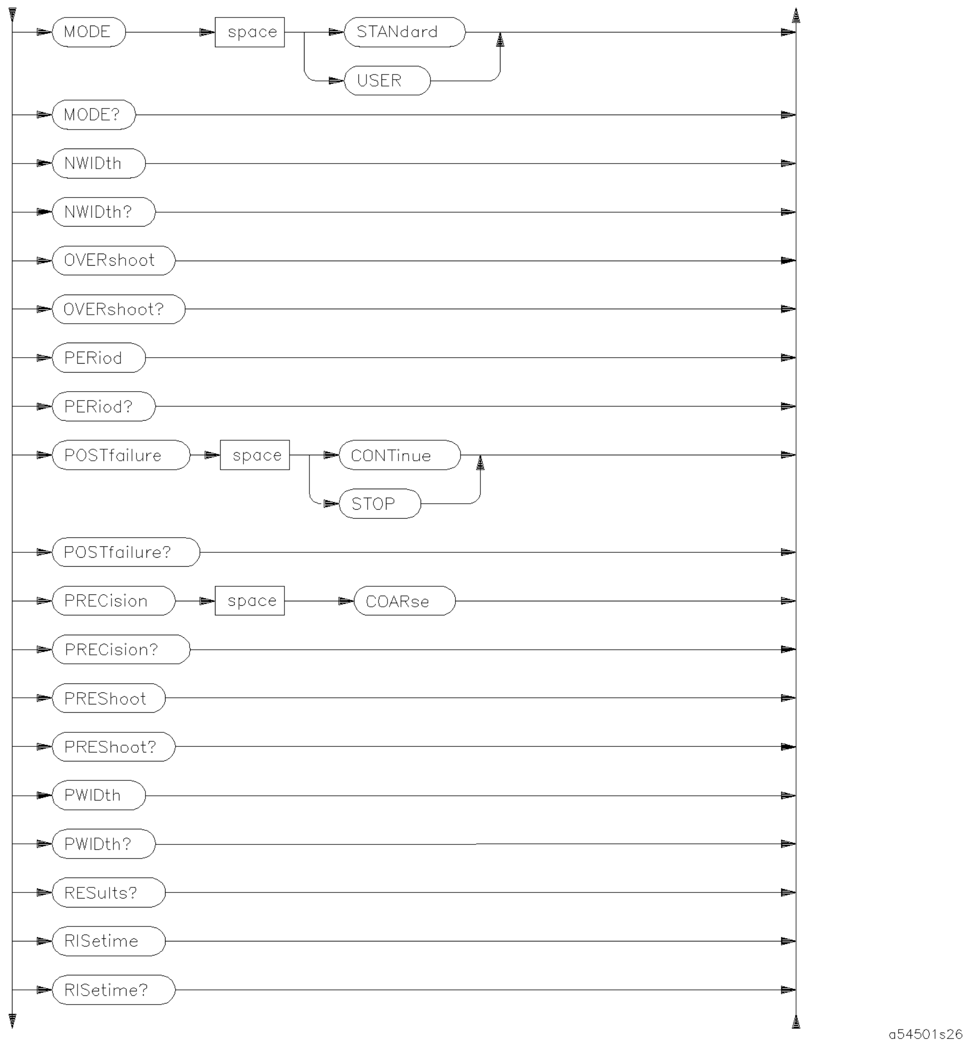


Figure 13-1. MEASURE Subsystem Commands Syntax Diagram (continued)

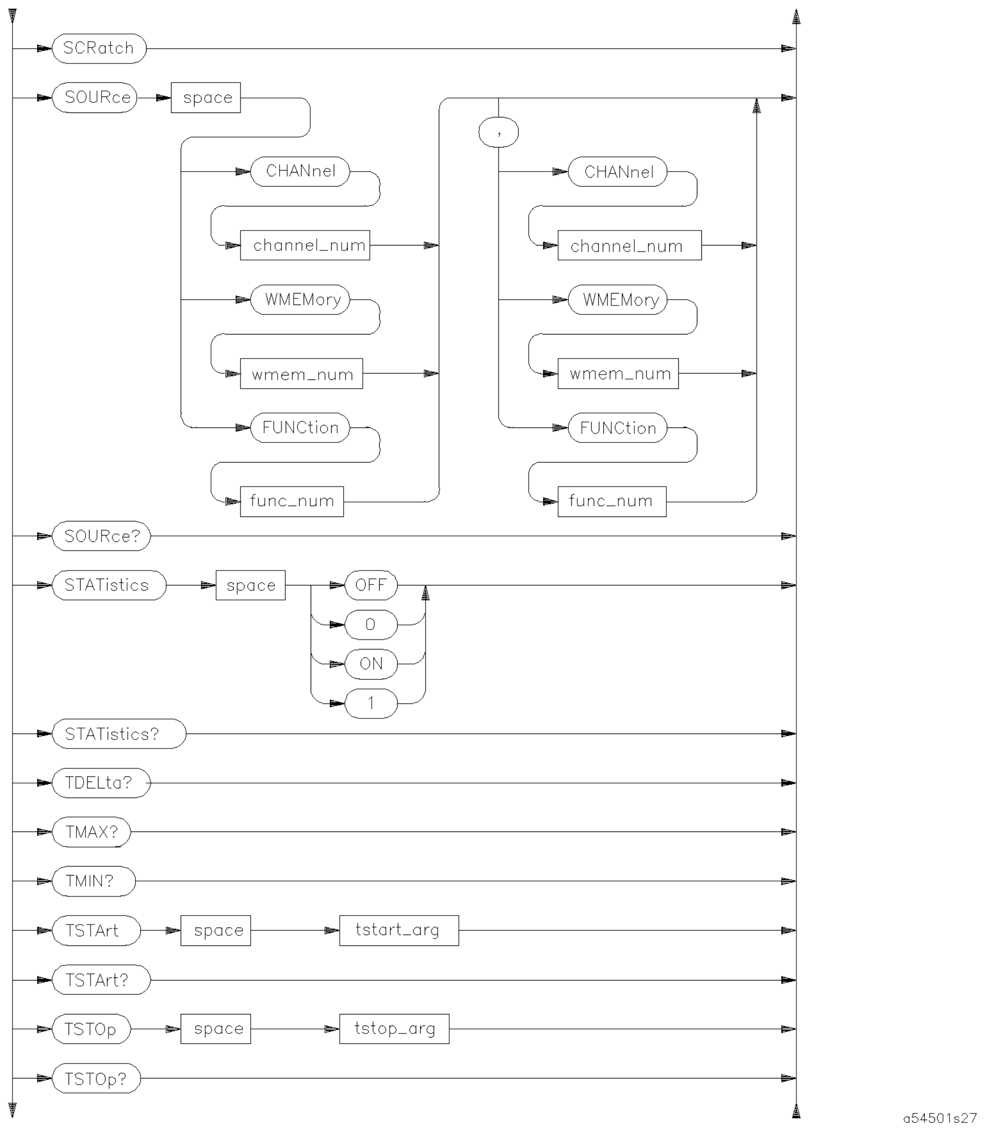


Figure 13-1. MEASURE Subsystem Commands Syntax Diagram (continued)

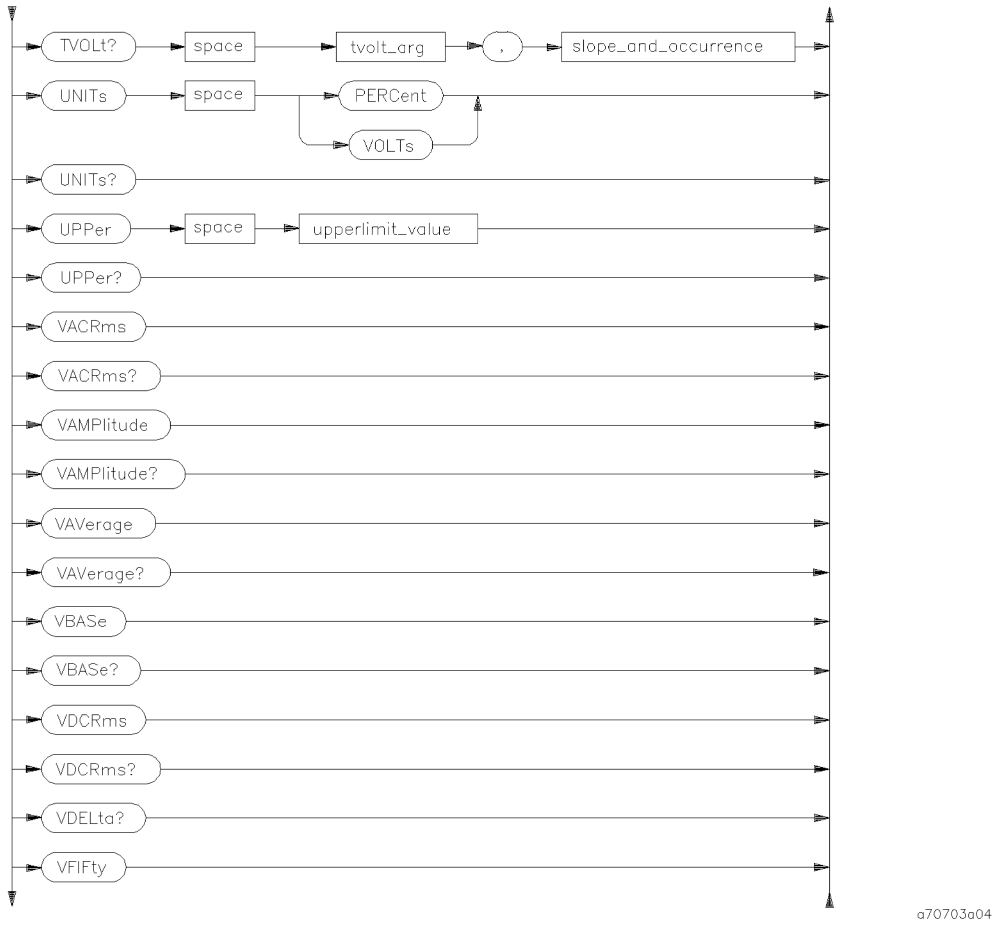
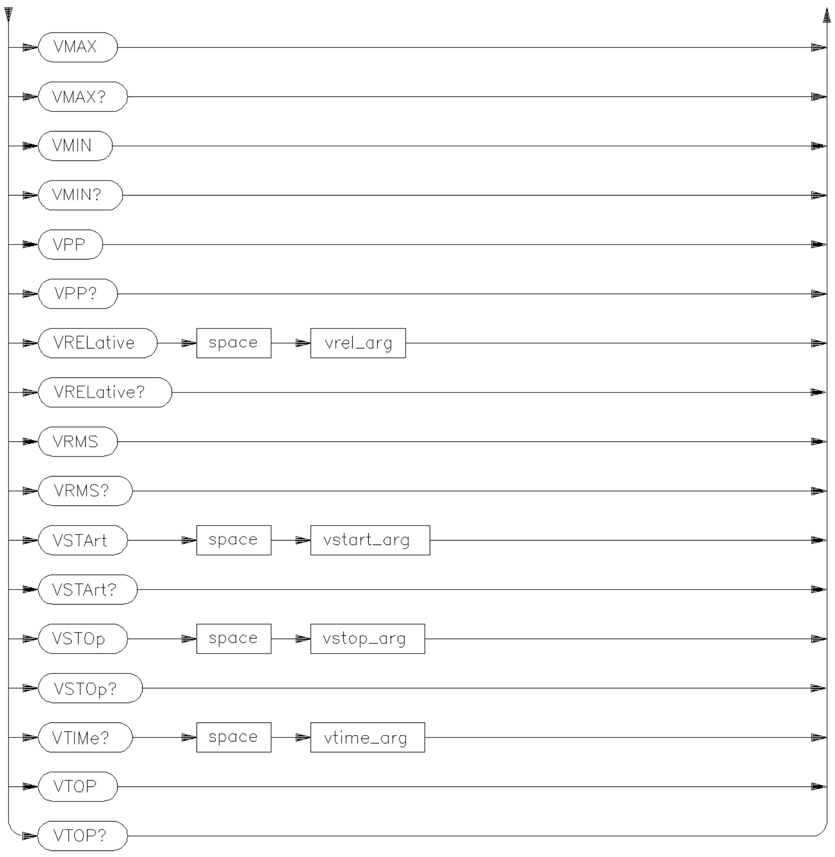


Figure 13-1. MEASURE Subsystem Commands Syntax Diagram (continued)



channel_num	=	integer 1 through 4
edge_num	=	integer, 1 through 127
func_num	=	integer, 1 or 2
level	=	MIDDle, UPPer, or LOWer
lower_limit	=	lower limit for compare
lowlimit_value	=	lower threshold value in percent or volts
measurement	=	name of measurement to be compared
polarity	=	positive or negative
slope_and_occurrence	=	integer, −127 to 127 (excluding 0) specifying a displayed edge
tstart_arg	=	time in seconds from trigger
tstop_arg	=	time in seconds from trigger
tvolt_arg	=	real number specifying voltage
upper_limit	=	upper limit for compare
upperlimit_value	=	upper threshold value in percent or volts
vrel_arg	=	integer, 0 to 100
vstart_arg	=	real number within voltage range
vstop_arg	=	real number within voltage range
vtime_arg	=	real number in the horizontal display window
wmem_num	=	integer, 1 through 4

a70703a05

Figure 13-1. MEASURE Subsystem Commands Syntax Diagram (continued)

ALL

The :MEASURE:ALL query makes a set of measurements on the present signal and sends the measurement results to the output buffer. Refer to the individual commands for information on how the measurements are made and the returned format of the measurement results.

Query Syntax

```
   :MEASure:ALL?
```

Returned Format

```
   <frequency>;<period>;<pwidth>;<nwidth>;<risetime>;<falltime>;
   <vamplitude>;<vpp>;<preshoot>;<overshoot>;<dutycycle>;<vacrms>;
   <vmax>;<vmin>;<vtop>;<vbase>;<vaverage>;<vdcrms><NL>
```

Where:

```
   <result> ::= individual measurement results (exponential - NR3 format)
```

Example

```
   DIM A$[500]
   OUTPUT 707;":MEASURE:ALL?"
   ENTER 707;"A$"
   PRINT A$
```

COMPare

The :MEASURE:COMPARE command specifies the measurement and limits to be used for the measurement comparison (limit test). The first limit is the upper limit, the second is the lower.

This command does not start the test, but only sets the test parameters.

The COMPARE query returns the current specification.

Command Syntax

```
   :MEASure:COMPare <measurement>,<upper_limit>,<lower_limit>
```

Where:

```
   <measurement> ::= {RISetime|FALLtime|FREQuency|PERiod|
                     PWIDth|NWIDth|VAMPlitude|VBASe|VTOP|VPP|VAVerage|
                     VMAX|VMIN|VACRms|VRMS|DUTycycle|DELAY|VDCRms}

   <upper_limit> ::= High limit value
   <lower_limit> ::= Low limit value
```

Note When setting the limits for Frequency the suffix “HZ” can be used.

Example

```
OUTPUT 707;":MEASURE:COMPARE RISETIME,90,10"
```

Query Syntax

```
:MEASure:COMPare? <measurement>
```

Where:

```
<measurement> ::= {RISetime|FALLtime|FREQuency|PERiod|
                  PWIDth|NWIDth|VAMPlitude|VBASe|VTOP|VPP|VAVerage|
                  VMAX|VMIN|VACRms|DUTycycle|DELAY|VDCRms}
```

Returned Format

```
<measurement>,<upper_limit>,<lower_limit><NL>
```

Example

```
DIM Cmp$[50]
OUTPUT 707;"MEASure:COMPare? VPP"
ENTER 707;Cmp$
PRINT Cmp$
```

For example, the sequence required to do a limit test on frequency is:

```
OUTPUT 707;":MEASURE:FREQ"  !Select measurement
OUTPUT 707;":MEASURE:COMPARE FREQ,1000HZ,10HZ" !Set measurement limits
OUTPUT 707;":MEASURE:LIMITTEST MEASURE"  !Start test
```

Note The only way to see if a limit test failure has occurred over the bus is by checking if bit 3 of the status byte is set to a 1.

CURSor

The :MEASURE:CURSOR query returns the time and voltage values of the specified marker as an ordered pair of time/voltage values.

When the CURSOR query is sent, no measurement is made and the cursors are not moved.

If DELTA is specified:
the instrument returns the value of delta V and delta T.

If START is specified:
the positions of the start marker and VMarker 1 are returned.

If STOP is specified:
the positions of the stop marker and VMarker 2 are returned.

Query Syntax

```
:MEASure:CURsor? {DELTa|STARt|STOP}
```

Returned Format

```
<time>,<voltage><NL>
```

Where:

```
<time> ::=    delta time, start time or stop time
<voltage> ::= delta  voltage, VMarker 1 voltage or VMarker 2 voltage
```

Example

```
OUTPUT 707;":MEAS:SOURCE CHAN1"
OUTPUT 707;":MEAS:CURSOR? START"
ENTER 707;Tme,Vlt
PRINT Tme,Vlt
```

DEFine

The :MEASURE:DEFINE command sets up the definition for a measurement.

The DEFINE query returns the current setup.

Command Syntax

```
:MEASure:DEFine <measurement_spec>
```

Where:

```
<measurement_spec> ::= {DELay <polarity>,<edge_num>, <level>,
                       <polarity>, <edge_num>, <level> |PWIDth
                       MIDDle|UPPer|LOWer}|NWIDth{MIDDle|UPPer|LOWer}}
```

Where:

```
<polarity> ::= {POSitive|NEGative}
<edge_num> ::= an integer, -127 to 127 (excluding 0)
               specifying a displayed edge
<level> ::=   {MIDDle|UPPer|LOWer}
```

Example

```
OUTPUT 707;":MEAS:DEFINE DELAY,POSITIVE,1,UPPER,NEGATIVE,2,MIDDLE"
```

This example will set the parameters for a time measurement from the first positive edge at the upper threshold level to the second negative edge at the middle threshold. If one source is specified, both parameters apply to that signal. If two sources are specified the measurement is from the first positive edge on source 1 to the second negative edge on source 2.

Query Syntax

```
:MEASure:DEFine? {DELay|PWIDth|NWIDth}
```

Returned Format

```
<measurement_spec><NL>
```

Where:

```
<measurement_spec> ::= {DELay <polarity>,<edge_num>, <level>, <polarity>,
                       <edge_num>, <level>|PWIDth{MIDDle|UPPer|LOWer}|
                       NWIDth{MIDDle|UPPer|LOWer}}
```

Where:

```
<polarity> ::= {POSitive | NEGative}
<edge_num} ::= −127 to 127 (excluding 0) (integer - NR1 format)
<level> ::=   {MIDDle | UPPer | LOWer}
```

Example

```
DIM Dfn$[100]
OUTPUT 707;":MEASure:DEFine? DELay"
ENTER 707;Dfn$
PRINT Dfn$
```

DELay

The :MEASURE:DELAY command causes the instrument to determine the delay from the first specified edge on one source to the next specified edge on the same source, or to the first specified edge on another source.

One or two sources can be specified with the :MEASURE:SOURCE command.

The DELAY query returns the specified delay value.

Command Syntax

```
:MEASure:DELay
```

Example

```
OUTPUT 707;":MEAS:DEL"
```

Query Syntax

```
:MEASure:DELay?
```

Returned Format

```
<delay_value><NL>
```

Where:

```
<delay_value> ::= time value in seconds (exponential - NR3 format)
```

Example

```
DIM Dly$[50]
OUTPUT 707;":MEAS:DELAY?"
ENTER 707;Dly$
PRINT Dly$
```

DESTination

The :MEASURE:DESTINATION command specifies the destination to be used when a comparison violation is found.

If a waveform memory is specified, the memory is overwritten each time a violation is found.

The DESTINATION query returns the destination currently specified.

Note If waveform memories are used as the destination, the source must be set up separately using the :WAVEFORM:SOURCE command.

Command Syntax

```
:MEASure:DESTination {WMEMory{1|2|3|4}|OFF}
```

Example

```
OUTPUT 707;":MEAS:DEST WMEM2"
```

Query Syntax

```
:MEASure:DESTination?
```

Returned Format

```
{WMEMory{1|2|3|4}|OFF}<NL>
```

Example

```
DIM Dst$[50]
OUTPUT 707;":MEAS:DEST?"
ENTER 707;Dst$
PRINT Dst$
```

DUTycycle

The :MEASURE:DUTYCYCLE command places the instrument in the continuous measurement mode and starts a duty cycle measurement.

The DUTYCYCLE query measures and outputs the duty cycle of the signal specified by the SOURCE command. The value returned for duty cycle is the ratio of the positive pulse width to period.

The positive pulse width and the period of the specified signal are measured, then the duty cycle is calculated. The duty cycle is calculated with the following formula:

duty cycle = (+pulse width/period)$\times$100

Command Syntax

```
:MEASure:DUTycycle
```

Example

```
OUTPUT 707;":MEASURE:DUTYCYCLE"
```

Query Syntax

```
:MEASure:DUTycycle?
```

Returned Format

```
<value><NL>
```

Where:

```
<value> ::= ratio of + pulse width to period (exponential - NR3 format)
```

Example

```
DIM Dc$[50]
OUTPUT 707;":MEASURE:DUTYCYCLE?"
ENTER 707;Dc$
PRINT Dc$
```

ESTArt

The :MEASURE:ESTART command causes the instrument to position the start marker on the specified edge and slope of the waveform. All edges must be present and are counted from the first edge of the acquired data, not at the reference point. The start marker is positioned at the point where VMarker 1 (set using :MEASURE:VSTART command) intersects the waveform.

The ESTART query returns the currently specified edge and slope of the edge start marker.

Note The short form of this command does not follow the defined convention. The short form "EST" is the same for ESTART and ESTOP, so be careful not to send this form for the ESTART command. Sending "EST" will produce an error.

Command Syntax

```
:MEASure:ESTArt <edge>
```

Where:

```
<edge> ::= –127 to 127 excluding 0 (if a positive value is sent
           the + sign may be omitted or a space may be used)
```

Example

```
OUTPUT 707;":MEASURE:ESTART 2"
```

This example places the start marker at the second positive-going intersection of the waveform and VMarker 1.

Query Syntax

```
:MEASure:ESTArt?
```

Returned Format

```
<edge><NL>
```

Where:

```
<edge> ::= edge number (integer - NR1 format)
```

Example

```
OUTPUT 707;":MEAS:ESTART?"
ENTER 707;Estart
PRINT Estart
```

ESTOp

The :MEASURE:ESTOP command causes the instrument to position the stop marker on the specified edge and slope of the acquired waveform. All edges must be present and are counted from the first edge of the acquired data, not at the reference point. The stop marker is positioned at the point where VMarker 2 (set using :MEASURE:VSTOP command) intersects the waveform. The desired edge is specified by sending an integer value after the command name. If a positive integer is sent, the oscilloscope places the stop marker on a positive-going waveform edge. If a negative integer is sent, the stop marker is placed on a negative-going waveform edge.

If VMarker 2 does not intersect the waveform as specified, the error message “Edges required not found” is displayed.

The ESTOP query returns the edge and slope of the stop marker.

Note The short form of this command does not follow the defined convention. The short form “EST” is the same for ESTART and ESTOP, so be careful not to send this form for the ESTOP command. Sending “EST” will produce an error.

Command Syntax

```
:MEASure:ESTOp <edge>
```

Where:

```
<edge> ::= −127 to 127 excluding 0 (if a positive value is sent the
           + sign may be omitted or a space may  be used)
```

Example

```
OUTPUT 707;":MEAS:ESTOP -2"
```

This example places the stop marker at the second negative-going intersection of the waveform at VMarker 2.

Query Syntax

```
:MEASure:ESTOp?
```

Returned Format

```
<edge><NL>
```

Where:

```
<edge> ::= edge number (integer - NR1 format)
```

Example

```
OUTPUT 707;":MEASURE:ESTOP?"
ENTER 707;Estop
PRINT Estop
```

FALLtime

The :MEASURE:FALLTIME command places the instrument in the continuous measurement mode and starts a fall time measurement.

The FALLTIME query turns continuous measurement off, then measures and outputs the fall time of the signal present to the output buffer. The fall time is determined by measuring the time at the upper threshold of the falling edge then measuring the time at the lower threshold of the falling edge and calculating the fall time with the formula: fall time = time at lower threshold point - time at upper threshold point.

If the horizontal scaling is questionable when this measurement is made, bit 0 of the :SUMMARY:QUESTIONABLE:TIME register is set (refer to the SUMMARY subsystem).

Command Syntax

```
:MEASure:FALLtime
```

Example

```
OUTPUT 707;":MEAS:FALL"
```

Query Syntax

```
:MEASure:FALLtime?
```

Returned Format

```
<value><NL>
```

Where:

```
<value> ::= time in seconds between lower threshold and upper threshold
            voltage points (exponential - NR3 format)
```

Example

```
DIM Fll$[50]
OUTPUT 707;":MEASURE:FALLTIME?"
ENTER 707;Fll$
PRINT Fll$
```

FREQuency

The :MEASURE:FREQUENCY command places the instrument in the continuous measurement mode and starts a frequency measurement.

The FREQUENCY query turns continuous measurement off, performs a frequency measurement one time on the signal present, and then sends the measurement results to the output buffer. The method the instrument uses to determine frequency is to measure the time of the first complete cycle, then calculate frequency as follows:

The algorithm is:

```
if first edge of waveform is rising

then

frequency = 1/(time at second rising edge - time at first rising edge)

else

frequency = 1/(time at second falling edge - time at first falling edge)
```

Command Syntax

```
:MEASure:FREQuency
```

Example

```
OUTPUT 707;":MEASURE:FREQ"
```

Query Syntax

```
:MEASURE:FREQuency?
```

Returned Format

```
<value><NL>
```

Where:

```
<value> ::= frequency in Hertz (exponential - NR3 format)
```

Example

```
DIM Frq$[50]
OUTPUT 707;":MEASURE:FREQUENCY?"
ENTER 707;Frq$
PRINT Frq$
```

LIMittest

The :MEASURE:LIMITTEST command allows a limit test to be performed. If LIMITTEST is sent with the MEASURE parameter, then the instrument starts the test. If the OFF parameter is sent, the test is stopped.

The LTF (limit test failure) bit of the status byte will be set when a failure is found.

Command Syntax

```
:MEASure:LIMittest {MEASure|OFF}
```

Example

```
OUTPUT 707;":MEAS:LIM MEAS"
```

LOWer

The :MEASURE:LOWER command sets the lower measurement threshold. This command sends the value to the instrument. The value that is sent will be in the units selected with the UNITS command.

Note The measure UNITS should be set prior to sending this value.

The LOWER query returns the current setting of the lower measurement threshold.

Command Syntax

```
:MEASure:LOWer <lowlimit_value>
```

Where:

```
<lowlimit_value> ::= user defined lower threshold in percent or volts
```

Example

```
OUTPUT 707;":MEASURE:LOWER 47"
```

Query Syntax

```
:MEASure:LOWer?
```

Returned Format

```
<lowlimit_value><NL>
```

Where:

```
<lowlimit_value> ::= user defined lower threshold in percent or volts
```

Example

```
DIM Lwr$[50]
OUTPUT 707;":MEAS:LOW?"
ENTER 707;Lwr$
PRINT Lwr$
```

MODE

The :MEASURE:MODE command sets the measurement mode (definitions and thresholds).

The MODE query returns the current mode setting.

Command Syntax

```
:MEASure:MODE {STANdard|USER}
```

Example

```
OUTPUT 707;":MEAS:MODE STAN"
```

Query Syntax

```
:MEASure:MODE?
```

Returned Format

```
{STANdard|USER}<NL>
```

Example

```
DIM Md$[50]
OUTPUT 707;":MEASURE:MODE?"
ENTER 707;Md$
PRINT Md$
```

NWIDth

The :MEASURE:NWIDTH command places the instrument in the continuous measurement mode and starts a negative pulse width measurement.

The NWIDTH query turns continuous measurement mode off and measures and outputs the width of the first negative pulse of the waveform using the 50% levels with standard measurements selected.

If user defined measurements are selected, then the measurement is made at the mid threshold value.

The algorithm is:

```
if the first edge of waveform is rising

then

width = (time at second rising edge - time at first falling edge)

else

width = (time at first rising edge - time at first falling edge)
```

Command Syntax

```
:MEASure:NWIDth
```

Example

```
OUTPUT 707;":MEAS:NWIDTH"
```

Query Syntax

```
:MEASure:NWIDth?
```

Returned Format

```
<value><NL>
```

Where:

```
<value> ::= negative pulse width in seconds (exponential - NR3 format)
```

Example

```
DIM Nwd$[50]
OUTPUT 707;":MEASURE:NWIDTH?"
ENTER 707;Nwd$
PRINT Nwd$
```

OVERshoot

The :MEASURE:OVERSHOOT command places the instrument in the continuous measurement mode and starts an overshoot measurement. The OVERSHOOT query turns continuous measurement mode off and measures and outputs the overshoot of a selected signal. OVERSHOOT measures the first edge of the waveform with the following algorithm:

```
if the first edge of waveform is rising

then

overshoot = (Vmax - Vtop)/Vamplitude

else

overshoot = (Vbase - Vmin)/Vamplitude
```

Command Syntax

```
:MEASure:OVERshoot
```

Example

```
OUTPUT 707;":MEAS:OVER"
```

Query Syntax

```
:MEASure:OVERshoot?
```

Returned Format

```
<value><NL>
```

Where:

```
<value> ::= ratio of overshoot to Vamplitude (exponential - NR3 format)
```

Example

```
DIM Ovr$[50]
OUTPUT 707;":MEASURE:OVERSHOOT?"
ENTER 707;Ovr$
PRINT Ovr$
```

PERiod

The :MEASURE:PERIOD command places the instrument in the continuous measurement mode and starts a period measurement.

The PERIOD query turns continuous measurement mode off and measures and outputs the period of the first complete cycle of the waveform. The period is measured at the 50% point when standard measurements are selected and at the mid threshold voltage level of the waveform when user-defined measurements are selected.

The algorithm for this measurement is:

```
if the first edge of waveform is rising

then

period = (time at second rising edge - time at first rising edge)

else

period = (time at second falling edge - time at first falling edge)
```

Command Syntax

```
:MEASure:PERiod
```

Example

```
OUTPUT 707;":MEAS:PERIOD"
```

Query Syntax

```
:MEASure:PERiod?
```

Returned Format

```
<value><NL>
```

Where:

```
<value> ::= waveform period in seconds (exponential - NR3 format)
```

Example

```
DIM Prd$[50]
OUTPUT 707;":MEASURE:PERIOD?"
ENTER 707;Prd$
PRINT Prd$
```

POSTfailure

The :MEASURE:POSTFAILURE command specifies what will occur after a violation has been found by the limit test. If CONTINUE is selected, the instrument will continue to look for another violation. If STOP is selected the instrument will stop the limit test.

If CONTINUE is selected and a violation is found the violation will be written to the desired location. If a waveform memory is selected as the destination then all subsequent violations will overwrite the previous violation.

The POSTFAILURE query returns the current selection.

Command Syntax

```
:MEASure:POSTfailure {CONTinue|STOP}
```

Example

```
OUTPUT 707;":MEAS:POST CONT"
```

Query Syntax

```
:MEASure:POSTfailure?
```

Returned Format

```
{CONTinue|STOP}<NL>
```

Example

```
DIM Pf$[50]
OUTPUT 707;":MEASURE:POSTFAILURE?"
ENTER 707;Pf$
PRINT Pf$
```

PRECision

The :MEASURE:PRECISION command is included in the HP 70703A for compatibility with other HP instruments. It has no effect on the HP 70703A.

The PRECISION query always returns COARSE in the HP 70703A.

Command Syntax

```
:MEASure:PRECision COARse
```

Example

```
OUTPUT 707;":MEAS:PREC COARSE"
```

Query Syntax

```
:MEASure:PRECision?
```

Returned Format

```
COARse<NL>
```

Example

```
DIM Pc$[50]
OUTPUT 707;":MEAS:PRECISION?"
ENTER 707;Pc$
PRINT Pc$
```

PREShoot

The :MEASURE:PRESHOOT command places the instrument in the continuous measurement mode and starts a preshoot measurement. The PRESHOOT query turns continuous measurement mode off and measures and outputs the preshoot of the selected signal. Preshoot measures the first edge on the signal present with the following algorithm:

```
if the first edge of the waveform is rising

then

preshoot = (Vbase - Vmin)/Vamplitude

else

preshoot = (Vmax - Vtop)/Vamplitude
```

Command Syntax

```
:MEASure:PREShoot
```

Example

```
OUTPUT 707;":MEASURE:PRES"
```

Query Syntax

```
:MEASure:PREShoot?
```

Returned Format

```
<value><NL>
```

Where:

```
<value> ::= ratio of preshoot to Vamplitude (exponential - NR3 format)
```

Example

```
DIM Prs$[50]
OUTPUT 707;":MEASURE:PRESHOOT?"
ENTER 707;Prs$
PRINT Prs$
```

PWIDth

The :MEASURE:PWIDTH command places the instrument in the continuous measurement mode and starts a positive pulse width measurement.

The PWIDTH query measures and outputs the width of the first positive pulse of the signal present. Pulse width is measured at the 50% voltage level with standard measurements selected and at the mid-level threshold value with user-defined measurements selected. The algorithm for this measurement is:

```
if the first edge of waveform is falling

then

width = (time at second falling edge - time at first rising edge)

else

width = (time at first falling edge - time at first rising edge)
```

Command Syntax

```
:MEASure:PWIDth
```

Example

```
OUTPUT 707;":MEAS:PWIDTH"
```

Query Syntax

```
:MEASure:PWIDth?
```

Returned Format

```
<value><NL>
```

Where:

```
<value>  ::= width of positive pulse in seconds (exponential - NR3 format)
```

Example

```
DIM Pwd$[50]
OUTPUT 707;":MEASURE:PWIDTH?"
ENTER 707;Pwd$
PRINT Pwd$
```

RESults

The :MEASURE:RESULTS query tells the instrument to return the currently active measurements. If statistics are on the current, minimum, maximum, and average will be returned for each measurement. If the limit test is on and POSTFAILURE is set to CONTINUE then the pass ratio will be returned instead of the average.

If the number of measurements returned is 0 then no <measurement>s are returned.

Query Syntax

```
:MEASure:RESults?
```

Returned Format

```
<No. of Meas>[;<measurement>]...<NL>
```

Where:

```
<No. of Meas> ::= number of measurements displayed on the CRT, 0
    through 8 (integer - NR1 format)
<measurement> ::= measurement_name measurement_result
```

Example

```
DIM Mr$[100]
OUTPUT 707;":MEASURE:RESULTS?"
ENTER 707;Mr$
PRINT Mr$
```

RISetime

The :MEASURE:RISETIME command places the instrument in the continuous measurement mode and starts a rise time measurement.

The RISETIME query turns continuous measurement mode off and measures and outputs the rise time of the first rising (positive-going) edge of the waveform. For best measurement accuracy set the sweep speed as fast as possible. The rise time is determined by measuring the time at the lower threshold of the rising edge then the time at the upper threshold of the rising edge and calculating the rise time with the formula:

rise time = (time at upper threshold point - time at lower threshold point)

If the horizontal scaling is questionable when this measurement is made bit 0 of the :SUMMARY:QUESTIONABLE:TIME register is set (refer to the SUMMARY subsystem).

Command Syntax

```
:MEASure:RISetime
```

Example

```
OUTPUT 707;":MEAS:RIS"
```

Query Syntax

```
:MEASure:RISetime?
```

Returned Format

```
<value><NL>
```

Where:

```
<value> ::= rise time in seconds (exponential - NR3 format)
```

Example

```
DIM Rs$[50]
OUTPUT 707;":MEAS:RIS?"
ENTER 707;Rs$
PRINT Rs$
```

SCRatch

The :MEASURE:SCRATCH command clears the measurement results from the measurement queue.

Command Syntax

```
:MEASure:SCRatch
```

Example

```
OUTPUT 707;":MEASURE:SCRATCH"
```

SOURce

The :MEASURE:SOURCE command selects the source(s) for the measurements. The source(s) specified will become the source(s) for the MEASURE subsystem commands.

Two sources can be specified with this command. All measurements except DELAY are made on the first specified source. The DELAY measurement will use two sources if two have been specified. If only one source is specified the DELAY measurement will use that source for both of its parameters.

The SOURCE query returns the current source selection. If the specified sources are different both will be returned, otherwise one source will be returned.

Command Syntax

```
:MEASure:SOURce <source1>[,<source2>]
```

Where:

```
<source1> and <source2> ::= {CHANnel{1|2|3|4}|FUNCtion{1|2}|WMEMory{1|2|3|4}}
```

Example

```
OUTPUT 707;":MEASURE:SOURCE CHANNEL1, WMEMORY1"
```

Query Syntax

```
:MEASure:SOURce?
```

Returned Format

```
<source1>[,<source2>]<NL>
```

Where:

```
<source1> and <source2> ::= {CHANnel{1|2|3|4}|FUNCtion{1|2}|WMEMory{1|2|3|4}}
```

Example

```
DIM Src$[50]
OUTPUT 707;":MEAS:SOUR?"
ENTER 707;Src$
PRINT Src$
```

STATistics

The :MEASURE:STATISTICS command allows the statistics mode to be controlled. When this mode is on, and the measurements are in the continuous mode, the minimum, maximum, average, and current measurement will be placed in the measurement queue. To read the measurement results, execute the :MEASURE:RESULTS? query.

The STATISTICS query returns the current mode.

Note "Average" will be replaced by "pass ratio" when limit test is selected and "after failure" is set to continue. Pass ratio lists the percentage of times a certain test passed.

Command Syntax

```
:MEASure:STATistics {{ON|1}|{OFF|0}}
```

Example

```
OUTPUT 707;":MEASURE:STAT ON"
```

Query Syntax

```
:MEASure:STATistics?
```

Returned Format

```
{1|0}<NL>
```

Example

```
DIM Stt$[50]
OUTPUT 707;":MEASURE:STAT?"
ENTER 707;Stt$
PRINT Stt$
```

TDELta

The :MEASURE:TDELTA query returns the time difference between the start and stop time markers, that is:

Tdelta = Tstop − Tstart

where Tstart is the time at the start marker and Tstop is the time at the stop marker.

Query Syntax

```
:MEASure:TDELta?
```

Returned Format

```
<value><NL>
```

Where:

```
<value> ::= difference between start and stop markers
             (exponential - NR3 format)
```

Example

```
DIM Tdl$[50]
OUTPUT 707;":MEASURE:TDELTA?"
ENTER 707;Tdl$
PRINT Tdl$
```

TMAX

The :MEASURE:TMAX query returns the time at which the first maximum voltage occurred.

Query Syntax

```
:MEASure:TMAX?
```

Returned Format

```
<time at max voltage><NL>
```

Example

```
DIM Tmx$[50]
OUTPUT 707;":MEASURE:TMAX?"
ENTER 707;Tmx$
PRINT Tmx$
```

TMIN

The :MEASURE:TMIN query returns the time at which the first minimum voltage occurred.

Query Syntax

```
:MEASure:TMIN?
```

Returned Format

```
<time at min voltage><NL>
```

Example

```
DIM Tmn$[50]
OUTPUT 707;":MEAS:TMIN?"
ENTER 707;Tmn$
PRINT Tmn$
```

TSTArt

The :MEASURE:TSTART command moves the start marker to the specified time with respect to the trigger time. The TSTART query returns the time at the start marker.

Note The short form of this command does not follow the defined convention. The short form "TST" is the same for TSTart and TSTOp, so be careful not to send this form for the TSTART command. Sending "TST" will produce an error.

Command Syntax

```
:MEASure:TSTArt <start marker time>
```

Where:

```
<start marker time> ::= time at start marker in seconds
                        (exponential - NR3 format)
```

Example

```
OUTPUT 707;":MEASURE:TSTART 30E-9"
```

Query Syntax

```
:MEASure:TSTArt?
```

Returned Format

```
<value><NL>
```

Where:

```
<value> ::= time at start marker in second
            (exponential - NR3 format)
```

Example

```
DIM Tst$
OUTPUT 707;":MEASURE:TSTART?"
ENTER 707;Tst$
PRINT Tst$
```

TSTOp

The :MEASURE:TSTOP command moves the stop marker to the specified time with respect to the trigger time.

The TSTOP query returns the time at the stop marker.

Note The short form of this command does not follow the defined convention. The short form "TST" is the same for TSTArt and TSTOp, so be careful not to send this form for the TSTOP command. Sending "TST" will produce an error.

Command Syntax

```
:MEASure:TSTOp <stop marker time>
```

Where:

```
<stop marker time> ::= time at stop marker in seconds
```

Example

```
OUTPUT 707;":MEAS:TSTOP 40E-9"
```

Query Syntax

```
:MEASure:TSTOp?
```

Returned Format

```
<value><NL>
```

Where:

```
<value> ::= time at stop marker in seconds
            (exponential - NR3 format)
```

Example

```
DIM Tst$[50]
OUTPUT 707;":MEASURE:TSTOP?"
ENTER 707;Tst$
PRINT Tst$
```

TVOLt

When the :MEASURE:TVOLT query is sent, the selected source is searched for the defined voltage level and transition. The time interval between the trigger event and this defined occurrence is returned as the response to this query.

The <voltage> can be specified as a negative or positive voltage. To specify a negative voltage, use a minus (−) sign. The sign of <slope> selects a rising (+) or falling (−) edge.

The magnitude of <occurrence> defines the occurrence to be reported. For example, +3 will return the time for the third time the waveform crosses the specified voltage level in the positive direction. Once this voltage crossing is found, the HP 70703A will output the time at that crossing in seconds, with the trigger point (time zero) as the reference.

If the specified crossing cannot be found, the HP 70703A outputs +9.99999E+37. This will happen if the waveform does not cross the specified voltage, or if the waveform does not cross the specified voltage for the specified number of times in the specified direction.

Query Syntax

```
:MEASure:TVOLt?  <voltage>, <slope><occurrence>
```

Where:

```
<voltage> ::= voltage level the waveform must cross.
              This can be a positive or negative voltage.

<slope> ::= direction of waveform when <voltage> is
            crossed rising (sp or +) or falling (−)

<occurrence> ::= number of crossing to be reported (if one, first
                 crossing reported; if two, second crossing is
                 reported, and so on.)
```

Returned Format

```
<time><NL>
```

Where:

```
<time> ::= time in seconds of specified voltage crossing
           (exponential - NR3 format)
```

Example

```
DIM Tvlt$[50]
OUTPUT 707;"MEASURE:TVOLT? -.250,+3"
ENTER 707;Tvlt$
PRINT Tvlt$
```

UNITs

The :MEASURE:UNITS command sets the measurement threshold units when the user defined measurement mode is selected. The UNITS can be set to PERCENT or VOLTS.

The UNITS query returns the currently selected units.

Command Syntax

```
:MEASure:UNITs {PERCent|VOLTs}
```

Example

```
OUTPUT 707;":MEASURE:UNITS PERCENT"
```

Query Syntax

```
:MEASure:UNITs?
```

Returned Format

```
{PERCent|VOLTs}<NL>
```

Example

```
DIM Unt$[50]
OUTPUT 707;":MEASURE:UNITS?"
ENTER 707;Unt$
PRINT Unt$
```

UPPer

The :MEASURE:UPPER command sets the upper measurement threshold.

Note The measure UNITS should be set prior to sending this value.

The UPPER query returns the value of the upper measurement threshold.

Command Syntax

```
:MEASure:UPPer <value>
```

Where:

```
<value> ::= upper threshold value in percent or volts
```

Example

```
OUTPUT 707;":MEAS:UNIT PERCENT"
OUTPUT 707;":MEAS:UPPER 90"
```

Query Syntax

```
:MEASure:UPPer?
```

Returned Format

```
<upper_threshold value><NL>
```

Where:

```
<upper_threshold value> ::= upper threshold value in percent
                            or volts (exponential - NR3 format)
```

Example

```
DIM Upp$[50]
OUTPUT 707;":MEAS:UPP?"
ENTER 707;Upp$
PRINT Upp$
```

VACRms

The :MEASURE:VACRMS command sets the HP 70703A in the continuous measurement mode and starts an AC RMS voltage measurement. The VACRMS query turns continuous measurement mode off, performs an AC RMS voltage measurement one time on the signal present, and then sends the measurement results to the output buffer.

Command Syntax

```
:MEASure:VACRms
```

Example

```
OUTPUT 707;":MEAS:VACR"
```

Query Syntax

```
:MEASure:VACRms?
```

Returned Format

```
<value><NL>
```

Where:

```
<value> ::= rms voltage of displayed points
            (exponential - NR3 format)
```

Example

```
DIM Vacr$[50]
OUTPUT 707;":MEAS:VACR?"
ENTER 707;Vacr$
PRINT Vacr$
```

Note The :MEASURE:VACRMS? query is identical to the :MEASURE:VRMS? query. The ACRMS voltage measurement is made using the first cycle present. If a complete cycle is not present, the instrument calculates the AC RMS value of all currently acquired data points.

VAMPlitude

The :MEASURE:VAMPLITUDE command places the HP 70703A in the continuous measurement mode and starts an amplitude voltage measurement. The VAMPLITUDE query turns continuous measurement mode off and returns the difference between the top and base voltage of the displayed signal. The VAMPLITUDE value will not normally be the same as the Vp-p value if the input signal is a pulse.

The VAMPLITUDE value is calculated with the formula:

Vamplitude = Vtop − Vbase

Command Syntax

```
:MEASure:VAMPlitude
```

Example

```
OUTPUT 707;":MEAS:VAMP"
```

Query Syntax

```
:MEASure:VAMPlitude?
```

Returned Format

```
<value><NL>
```

Where:

```
<value> ::= difference between top and base voltages
            (exponential - NR3 format)
```

Example

```
DIM Vmp$[50]
OUTPUT 707;":MEASURE:VAMPLITUDE?"
ENTER 707;Vmp$
PRINT Vmp$
```

VAVerage

The :MEASURE:VAVERAGE command places the HP 70703A in the continuous measurement mode and starts an average voltage measurement. The VAVERAGE query turns continuous measurement mode off and calculates the average voltage over the waveform present.

Command Syntax

```
:MEASure:VAVerage
```

Example

```
OUTPUT 707;":MEAS:VAV"
```

Query Syntax

```
:MEASure:VAVerage?
```

Returned Format

```
<avg_value><NL>
```

Where:

```
<avg_value> ::= calculated average voltage
                (exponential - NR3 format)
```

Example

```
DIM Vv$[50]
OUTPUT 707;":MEAS:VAV?"
ENTER 707;Vv$
PRINT Vv$
```

Note Average voltage measurement is made using the first cycle present. If a complete cycle is not present, all currently acquired data points are averaged.

VBASe

The :MEASURE:VBASE command places the HP 70703A in the continuous measurement mode and starts a base voltage measurement.

The VBASE query turns continuous measurement mode off and measures and outputs the voltage value at the base of the waveform. The base voltage of a pulse is normally not the same as the minimum value.

Command Syntax

```
:MEASure:VBASe
```

Example

```
OUTPUT 707;":MEAS:VBASE"
```

Query Syntax

```
:MEASure:VBASe?
```

Returned Format

```
<value><NL>
```

Where:

```
<value> ::= voltage at base of selected waveform
                (exponential - NR3 format)
```

Example

```
DIM Vbs$[50]
OUTPUT 707;":MEASURE:VBASE?"
ENTER 707;Vbs$
PRINT Vbs$
```

VDCRms

The :MEASURE:VDCRMS command sets the HP 70703A in the continuous measurement mode and starts a DC RMS voltage measurement.

The VDCRMS query turns continuous measurement mode off, performs a DC RMS voltage measurement one time on the signal present, and then sends the measurement results to the output buffer.

Command Syntax

```
:MEASure:VDCRms
```

Example

```
OUTPUT 707;":MEAS:VDCR"
```

Query Syntax

```
:MEASure:VDCRms?
```

Returned Format

```
<value><NL>
```

Where:

```
<value> ::= rms voltage of displayed points (exponential - NR3 format)
```

Example

```
DIM Vdcr$[50]
OUTPUT 707;":MEAS:VDCR?"
ENTER 707;Vdcr$
PRINT Vdcr$
```

Note The DC RMS voltage measurement is made using the first cycle present. If a complete cycle is not present, the instrument calculates the DC RMS value of all currently acquired data points.

VDELta

The :MEASURE:VDELTA query outputs the voltage difference between the start (VMarker 1) and stop (VMarker 2) voltage markers. No measurement is made when the VDELTA query is received by the HP 70703A. The delta voltage value that is output is the current value.

VDELTA = Voltage at VMarker 2 − Voltage at VMarker 1

Query Syntax

```
:MEASure:VDELta?
```

Returned Format

```
<value><NL>
```

Where:

```
<value> ::= delta V value in volts (exponential - NR3 format)
```

Example

```
DIM Vdl$[50]
OUTPUT 707;":MEAS:VDELTA?"
ENTER 707;Vdl$
PRINT Vdl$
```

VFIFty

The :MEASURE:VFIFTY command instructs the HP 70703A to find the top and base values of the specified waveform(s), then places the voltage markers at the 50% voltage point on the specified source(s).

If only one source has been specified with the source command, the VFIFTY command sets both voltage markers (VMarker 1 and VMarker 2) to the 50% voltage level on that source.

If two sources are specified with the source command, VMarker 1 is set to the 50% level of the first specified source and VMarker 2 is set to the 50% level of the second specified source.

There is no query form of this command.

Command Syntax

```
:MEASure:VFIFty
```

Example

```
OUTPUT 707;":MEASURE:VFIFTY"
```

VMAX

The :MEASURE:VMAX command places the HP 70703A in the continuous measurement mode and starts a maximum voltage measurement.

The VMAX query turns continuous measurement mode off and measures and outputs the absolute maximum voltage present on the selected waveform.

Command Syntax

```
:MEASure:VMAX
```

Example

```
OUTPUT 707;":MEAS:VMAX"
```

Query Syntax

```
:MEASure:VMAX?
```

Returned Format

```
<value><NL>
```

Where:

```
<value> ::= maximum voltage of selected waveform (exponential - NR3 format)
```

Example

```
DIM Vmx$[50]
OUTPUT 707;":MEASURE:VMAX?"
ENTER 707;Vmx$
PRINT Vmx$
```

VMIN

The :MEASURE:VMIN command places the HP 70703A in the continuous measurement mode and starts a minimum voltage measurement. The VMIN query turns continuous measurement mode off and measures and outputs the absolute minimum voltage present on the selected waveform.

Command Syntax

```
:MEASure:VMIN
```

Example

```
OUTPUT 707;":MEAS:VMIN"
```

Query Syntax

```
:MEASure:VMIN?
```

Returned Format

```
<value><NL>
```

Where:

```
<value> ::= minimum voltage value of the selected waveform
            (exponential - NR3 format)
```

Example

```
DIM Vmn$[50]
OUTPUT 707;":MEASURE:VMIN?"
ENTER 707;Vmn$
PRINT Vmn$
```

VPP

The :MEASURE:VPP command places the HP 70703A in the continuous measurement mode and starts a peak-to-peak voltage measurement. The VPP query turns continuous measurement mode off, measures the maximum and minimum voltages for the selected source, then calculates the peak-to-peak voltage and outputs that value. The peak-to-peak voltage (Vpp) is calculated with the formula:

Vpp = Vmax − Vmin

where Vmax and Vmin are the maximum and minimum voltages present on the selected source.

Command Syntax

```
:MEASure:VPP
```

Example

```
OUTPUT 707;":MEAS:VPP"
```

Query Syntax

```
:MEASure:VPP?
```

Returned Format

```
<value><NL>
```

Where:

```
<value> ::= voltage peak to peak (exponential - NR3 format)
```

Example

```
DIM Vp$[50]
OUTPUT 707;":MEAS:VPP?"
ENTER 707;Vp$
PRINT Vp$
```

VRELative

The :MEASURE:VRELATIVE command moves the voltage markers to the specified percentage points of their last established position. The last established position is not necessarily on the waveform currently presented.

For example, after a :MEAS:VAMPLITUDE? query has been sent VMarker 1 is located at the base (0%) of the signal and VMarker 2 is at the top (100%) of the signal. If the VRELATIVE 10 command was executed, VMarker 1 is moved to the 10% level and VMarker 2 to the 90% level of the signal.

Any value between 0% and 100% can be used. If VRELATIVE 0 is sent, the markers are not moved, because the command indicates 0% movement from the current position.

As an example, when the following values are sent, the markers are moved to the following percentage values of their current position.

10 moves VMarker 1 to 10% and VMarker 2 to 90%

20 moves VMarker 1 to 20% and VMarker 2 to 80%

50 moves both markers to 50%

80 moves Vmarker 1 to 20% and Vmarker 2 to 80%

90 moves VMarker 1 to 10% and VMarker 2 to 90%

The starting position of the markers must be known for this command to be meaningful. The markers can be set to a known position on the selected waveform using the :MEAS:VAMPLITUDE? query.

The VRELATIVE query returns the current relative position of VMarker2 which is always in the range 50% through 90%.

Note The VRELATIVE command does not affect the upper and lower thresholds selected by the UPPER and LOWER commands.

Command Syntax

```
:MEASure:VRELative <percent>
```

Where:

```
<percent> ::= {0 through 100}
```

Example

```
OUTPUT 707;":MEASURE:VRELATIVE 20"
```

Query Syntax

```
:MEASure:VRELative?
```

Returned Format

```
<value><NL>
```

Where:

```
<value> ::= Vmarker 2 relative position in percent {50 through 100}
            (integer - NR1 format)
```

Example

```
DIM Vrl$[50]
OUTPUT 707;":MEAS:VREL?"
ENTER 707;Vrl$
PRINT Vrl$
```

VRMS

The :MEASURE:VRMS command places the HP 70703A in the continuous measurement mode and starts a RMS voltage measurement.

The VRMS query turns continuous measurement mode off and measures and outputs the RMS voltage of the selected waveform.

Note The :MEASURE:VRMS? query is identical to the :MEASURE:VACRMS? query. The AC RMS voltage measurement is made using the first cycle present. If a complete cycle is not present, the instrument calculates the AC RMS value of all currently acquired data points.

Command Syntax

```
:MEASure:VRMS
```

Example

```
OUTPUT 707;":MEAS:VRMS"
```

Query Syntax

```
:MEASure:VRMS?
```

Returned Format

```
<value><NL>
```

Where:

```
<value> ::= rms voltage of displayed points (exponential - NR3 format)
```

Example

```
DIM Vrm$[50]
OUTPUT 707;":MEAS:VRMS?"
ENTER 707;Vrm$
PRINT Vrm$
```

VSTArt

The :MEASURE:VSTART command moves VMarker 1 to the specified voltage. The values are limited to the currently defined channel, function, or memory range.

The VSTART query returns the current voltage level of VMarker 1.

Note The short form of this command does not follow the defined convention. The short form "VST" is the same for VSTART and VSTOP, so be careful not to send this form for the VSTART command. Sending "VST" will produce an error.

Command Syntax

```
:MEASure:VSTArt <voltage>
```

Where:

```
<voltage> ::= voltage value for VMarker 1
```

Example

```
OUTPUT 707;":MEAS:VSTA -10MV"
```

Query Syntax

```
:MEASure:VSTArt?
```

Returned Format

```
<value><NL>
```

Where:

```
<value> ::= voltage value for VMarker 1
```

Example

```
DIM Vst$
OUTPUT 707;":MEASURE:VSTART?"
ENTER 707;Vst$
PRINT Vst$
```

VSTOp

The :MEASURE:VSTOP command moves VMarker 2 to the specified voltage. The values are limited to the currently defined channel, function, or memory range.

The VSTOP query returns the current voltage level of VMarker 2.

Note The short form of this command does not follow the defined convention. The short form "VST" is the same for VSTART and VSTOP, so be careful not to send this form for the VSTOP command. Sending "VST" will produce an error.

Command Syntax

```
:MEASure:VSTOp <voltage>
```

Where:

```
<voltage> ::= voltage value for VMarker 2
```

Example

```
OUTPUT 707;":MEAS:VSTO -100MV"
```

Query Syntax

```
:MEASure:VSTOp?
```

Returned Format

```
<value><NL>
```

Where:

```
<value> ::= voltage value for VMarker 2
```

Example

```
DIM Vst$
OUTPUT 707;":MEASURE:VSTOP?"
ENTER 707;Vst$
PRINT Vst$
```

VTIMe

The :MEASURE:VTIME query returns the voltage at a specified time. The time is referenced to the trigger event and must be on the acquired waveform.

Query Syntax

```
:MEASure:VTIMe? <time>
```

Where:

```
<time> ::= displayed time from trigger in seconds
```

Returned Format

```
<value><NL>
```

Where:

```
<value> ::= voltage at specified time (exponential - NR3 format)
```

Example

```
DIM Vtm$[50]
OUTPUT 707;":MEAS:VTIM? .001"
ENTER 707;Vtm$
PRINT Vtm$
```

VTOP

The :MEASURE:VTOP command places the HP 70703A in the continuous measurement mode and starts a top voltage measurement. The VTOP query turns continuous measurement mode off and measures and outputs the voltage at the top of the waveform.

Command Syntax

```
:MEASure:VTOP
```

Example

```
OUTPUT 707;":MEAS:VTOP"
```

Query Syntax

```
:MEASure:VTOP?
```

Returned Format

```
<value><NL>
```

Where:

```
<value> ::= voltage at top of waveform (exponential - NR3 format)
```

Example

```
DIM Vtp$[50]
OUTPUT 707;":MEASURE:VTOP?"
ENTER 707;Vtp$
PRINT Vtp$
```

Summary Subsystem

The SUMMARY subsystem is used to examine the status of the oscilloscope calibration and self-test results by monitoring (reading the bit value) the various register groups. Figure 14-1 shows the SUMMARY registers in the oscilloscope.

Standard Event Status Register (*ESE) operates under IEEE488.2 control (see Common Commands for more information on this register).

Status Byte Register (*STB?) operates under IEEE488.2 control (see Common Commands for more information on this register).

Trigger Register (TER?) operates under oscilloscope control (see Root Level Commands for more information on this register).

Limit Test Register (LTER?) operates under oscilloscope control (see Root Level Commands for more information on this register).

Questionable Data/Signal Register operates under oscilloscope control.

The illustration shown in figure 14-5 illustrates the Questionable Data/Signal Register.

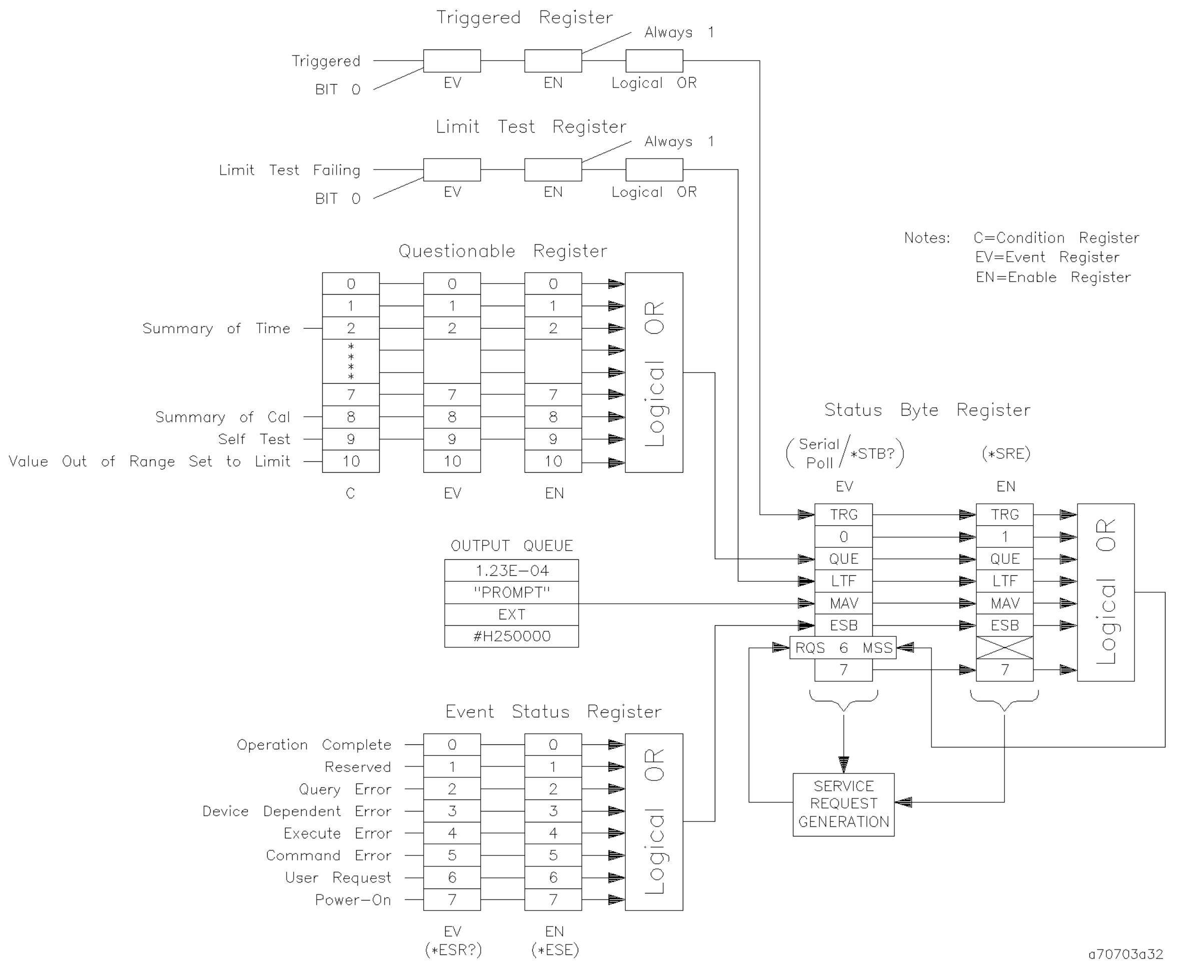


Figure 14-1. Oscilloscope Summary Registers

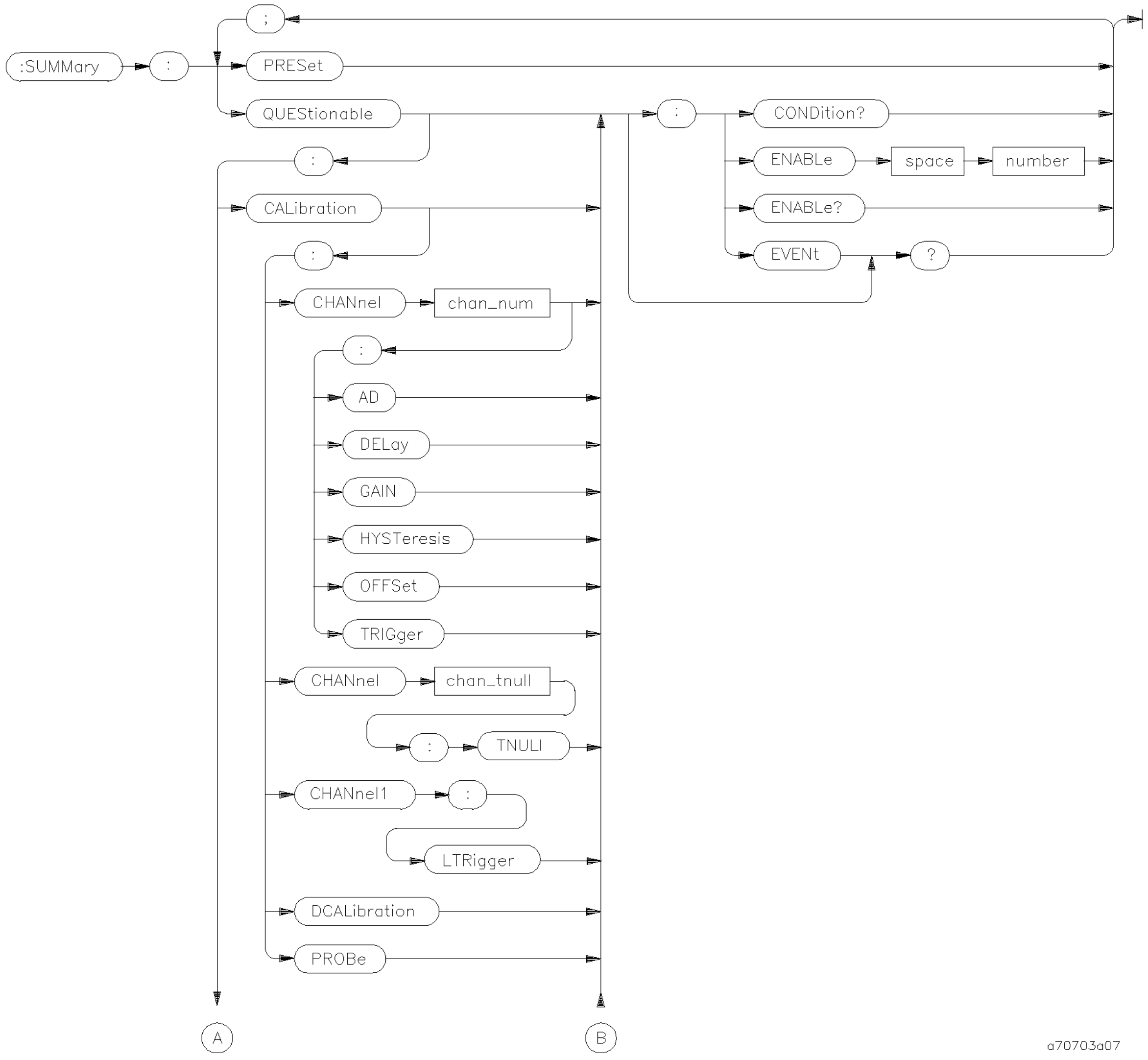
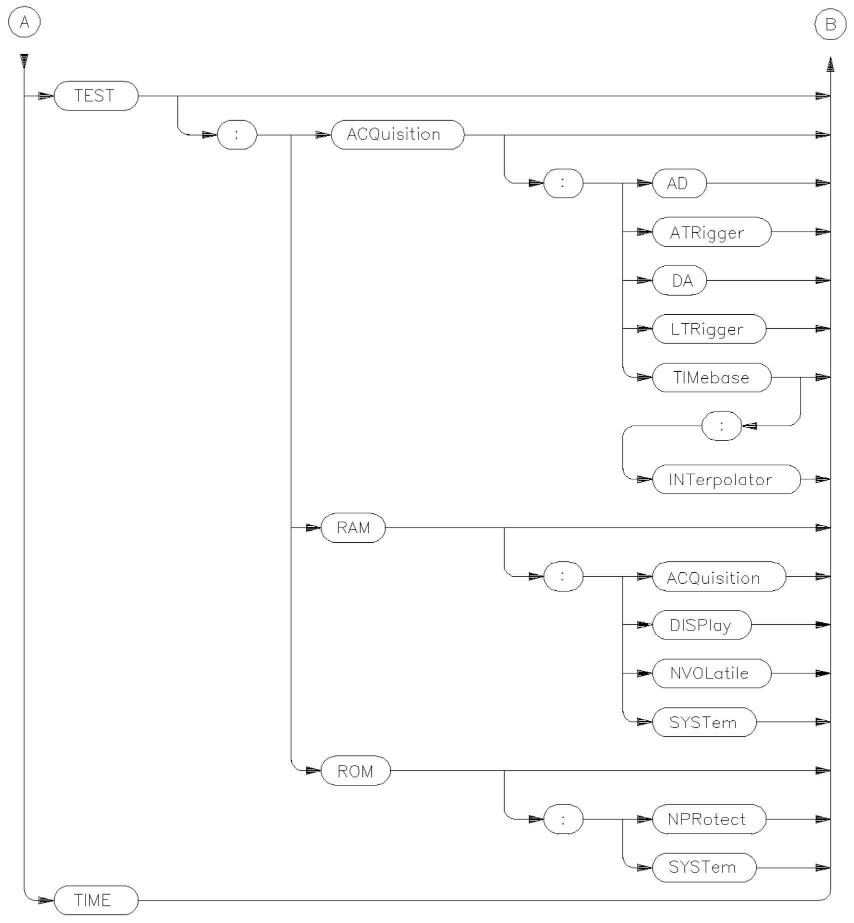


Figure 14-2. SUMMARY Subsystem Commands Syntax Diagram



chan_num = integer, 1 through 4
chan_tnull = integer, 2 through 4
number = decimal weighted bit value (see figure 14-3, 2 of 2)

a70703a08

Figure 14-2. SUMMARY Subsystem Commands Syntax Diagram (continued)

PRESet

The :SUMMARY:PRESET command sets the contents of the HP 70703A enable registers to a known state. When executed, the PRESET command affects all QUESTIONABLE ENABLE registers, and sets all bits true (1).

Command Syntax

```
:SUMMary:PRESet
```

Example

```
OUTPUT 707;":SUMM:PRES"
```

Note PRESET does not affect the Status Byte or Event Status registers. The Triggered and Limit Test ENABLE registers are always set to 1. The QUESTIONABLE ENABLE register is set to 0. PRESET does not affect any of the QUESTIONABLE EVENT registers. Use the *CLS command to clear all event registers.

QUEStionable

The :SUMMARY:QUESTIONABLE subsystem contains 52 separate registers that, through summing registers, eventually report to the QUESTIONABLE Data/Signal register. See figure 14-5 and the syntax diagram at the beginning of this section for a list of all the registers that set the QUESTIONABLE Data/Signal register.

A diagram is provided for each register in the QUESTIONABLE Data/Signal Register System as shown in figure 14-3. The following description for using the CONDITION?, [:EVENT]?, and ENABLE commands applies to all registers within the oscilloscope.

These registers are set and queried using decimal weighted bit values. The decimal equivalent for bits 0 to 15 is shown in figure 14-4. As an example, sending a decimal value of 4608 will set bits 9 and 12 true (1).

Each individual register (XXXXX) in the oscilloscope is made up of three separate registers:

:CONDITION register
:EVENT register
:ENABLE register

:CONDition?

:SUMMARY:QUESTIONABLE:XXXXX:CONDITION? queries the current contents of the specified (XXXXX):CONDITION register. The contents of all the Condition Registers are always set to “0”.

Example

```
:SUMMary:QUEStionable:XXXXX:CONDition?
```

This example queries the specified (XXXXX) Condition Register.

:ENABle

:SUMMARY:QUESTIONABLE:XXXXX:ENABLE <number> sets the enable mask, which allows true conditions (transitions) in the specified (XXXXX):EVENT register (bits 0 to 14) to be reported.

Example

```
:SUMMary:QUEStionable:XXXXX:ENABle 7680
```

This example set bits 9 to 12 true.

:ENABle?

:SUMMARY QUESTIONABLE:XXXXX:ENABLE returns the bit value of the specified (XXXXX):ENABLE register. Returns a decimal weighted value from 0 to 32767 indicating which bits are set true. Reading the Enable Register does not clear its contents.

Example

```
:SUMMary:QUEStionable:XXXXX:ENABle?
```

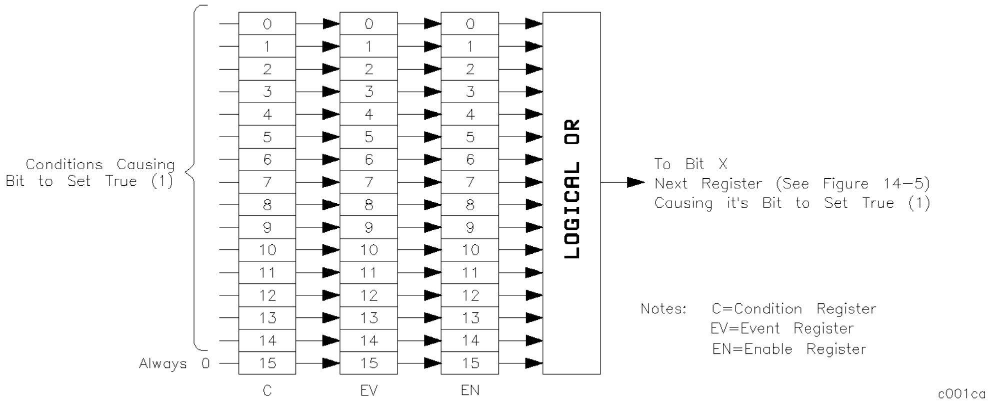
This example queries the specified (XXXXX) Enable Register, without clearing the contents.

[:EVENt]?

:SUMMARY:QUESTIONABLE:XXXXX[:EVENT] queries the status of the specified (XXXXX):EVENT Register. The Event Register latches only low to high events from the specified (XXXXX):CONDITION Register. Returns a decimal weighted value from 0 to 32767 indicating which bits are set true. Reading the specified event register by a query will clear its contents.

```
:SUMMary:QUEStionable:XXXXX?
```

This example queries the specified (XXXXX) event register and clears the contents.



Conditions Causing Bit to Set True (1)
Always 0
0
1
2
3
4
5
6
7
8
9
10
11
12
13
14
15
C
EV
EN
LOGICAL OR
To Bit X Next Register (See Figure 14-5) Causing it's Bit to Set True (1)
Notes: C=Condition Register
EV=Event Register
EN=Enable Register
c001ca

Figure 14-3. Specified (XXXXX) Registers

Bit Number	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
Decimal Value	1	2	4	8	16	32	64	128	256	512	1024	2048	4096	8192	16384	reserved

c001c

Figure 14-4. Bit Number to Decimal Value

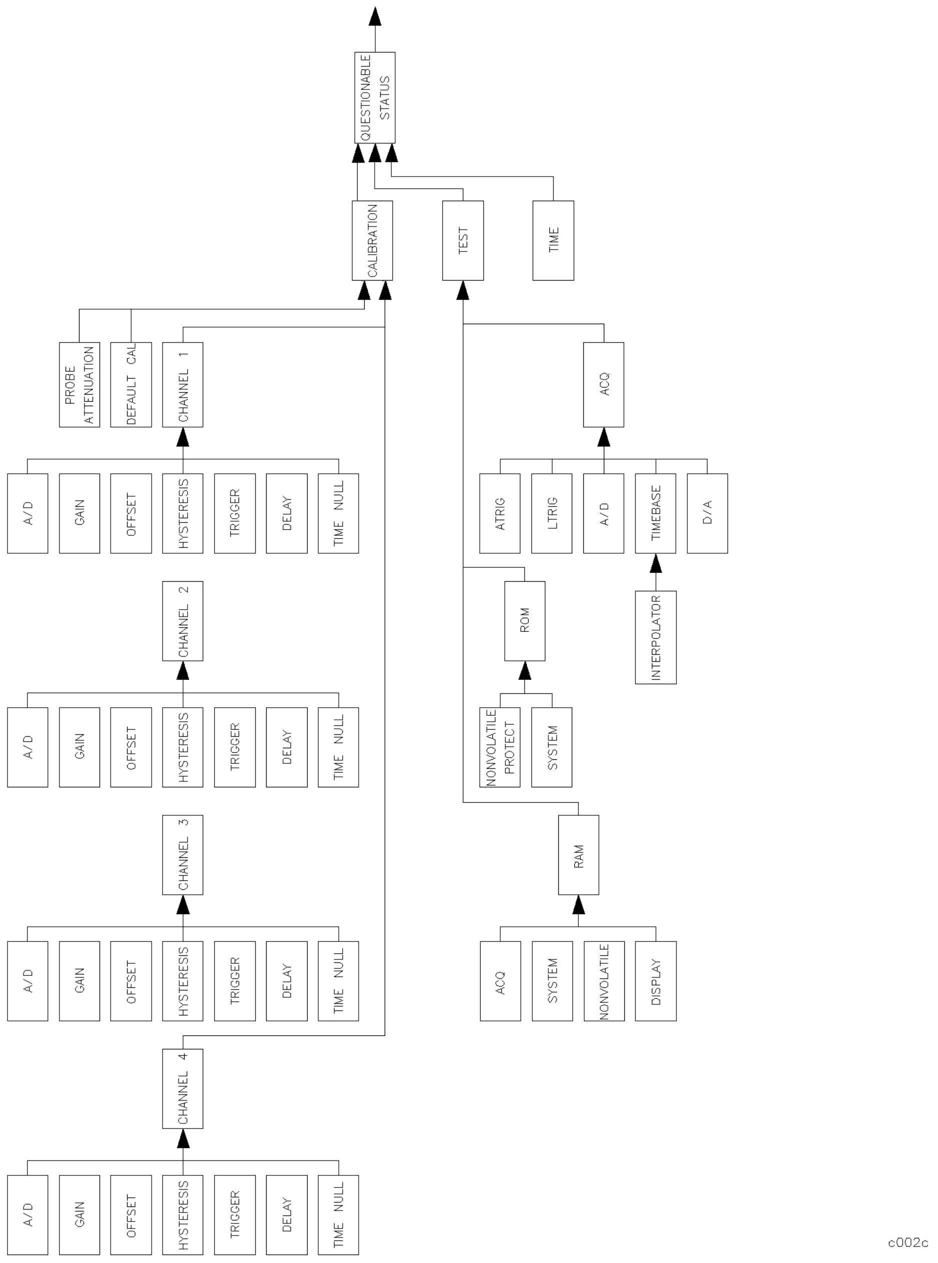


Figure 14-5. Summary Questionable Data/Signal Register Subsystem

QUEStionable:CALibration

The :SUMMARY:QUESTIONABLE:CALIBRATION register reports a summary of calibration results and status for all channels to the Questionable Data/Signal Register. Use the diagram to interpret returned results. See figures 14-3, 14-4, and 14-5 for additional information on using the CONDITION?, ENABLE, ENABLE?, and [:EVENT]? commands.

Command Syntax

```
:SUMMary:QUEStionable:CALibration:ENABle <number>
```

Example

```
:SUMM:QUES:CAL:ENAB 4608
```

The example shown sets the enable mask, which allows true conditions (transitions) in the CALIBRATION:EVENT register to be reported. In this example, calibration enable register bits 9 and 12 are set to true (1).

Query Syntax

```
:SUMMary:QUEStionable:CALibration:{:CONDition?|:ENABle?|[:EVENt]?}
```

Example

```
:SUMM:QUES:CAL:ENAB?
```

The example shown queries the instrument to return the specified register contents.

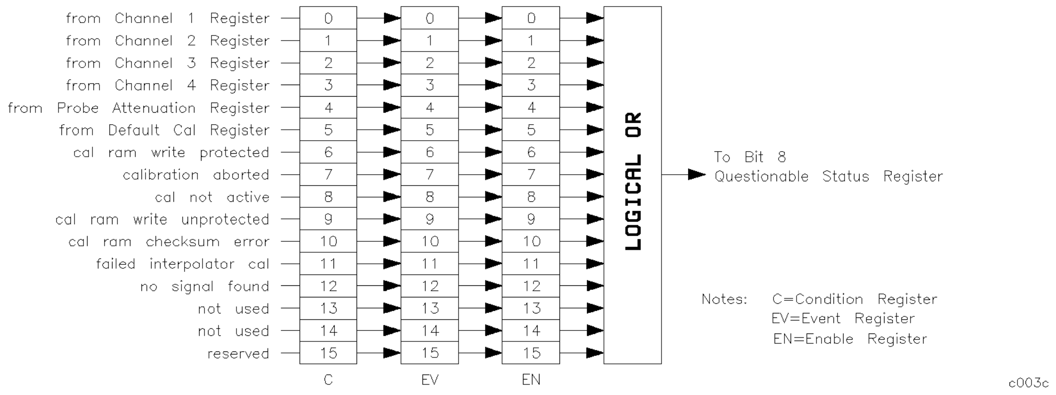


Figure 14-6. Summary of Calibration Register

QUEStionable:CALibration:CHANnel<N>

The :SUMMARY:QUESTIONABLE:CALIBRATION:CHANNEL<N> register reports the status of calibration data for the specified channel (1 to 4). Use the following diagram to interpret the returned results. See figures 14-3, 14-4, and 14-5 for additional information on using the CONDITION?, ENABLE, ENABLE?, and [:EVENT]? commands.

Command Syntax

```
:SUMMary:QUEStionable:CALibration:CHANnel{1|2|3|4}:ENABle<number>
```

Example

```
:SUMM:QUES:CAL:CHAN3:ENAB 4608
```

The example shown sets the enable mask, which alrows true conditions (transitions) in the CALIBRATION:CHANNEL3:EVENT Register to be reported. In this example, channel 3 calibration enable register bits 9 and 12 are set to true (1).

Query Syntax

```
:SUMMary:QUEStionable:CALibration:CHANnel{1|2|3|4}{:CONDition?|ENABle?|[:EVENt]?}
```

Example

```
:SUMM:QUES:CAL:CHAN2:ENAB?
```

The example shown queries the instrument to return the specified register contents.

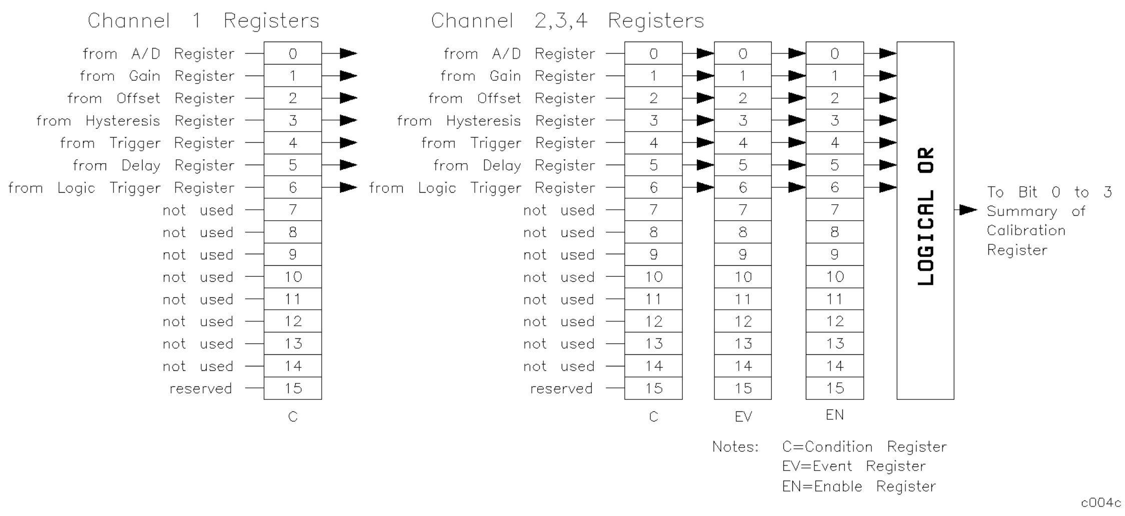


Figure 14-7. Channel Registers

QUEStionable:CALibration:CHANnel<N>:AD

The :SUMMARY:QUESTIONABLE:CALIBRATION:CHANNEL<N>:AD register reports the status of the A/D calibration data for the specified channel (1 to 4). Use the following diagram to interpret the returned results. See figures 14-3, 14-4, and 14-5 for additional information on using the CONDITION?, ENABLE, ENABLE?, and [:EVENT]? commands.

Command Syntax

```
:SUMMary:QUEStionable:CALibration:CHANnel{1|2|3|4}:AD:ENABle <number>
```

Example

```
:SUMM:QUES:CAL:CHAN1:AD:ENAB 112
```

The example shown sets the enable mask, which allows true conditions (transitions) in the :CALIBRATION:CHANNEL1:AD:EVENT register to be reported. In this example, channel 1 calibration A/D enable register bits 4 through 6 are set to true (1).

Query Syntax

```
:SUMMary:QUEStionable:CALibration:CHANnel{1|2|3|4}:AD
{:CONDition?|:ENABle?|[:EVENt]?}
```

Example

```
:SUMM:QUES:CAL:CHAN3:AD?
```

The example shown queries the instrument to return the contents of the specified register.

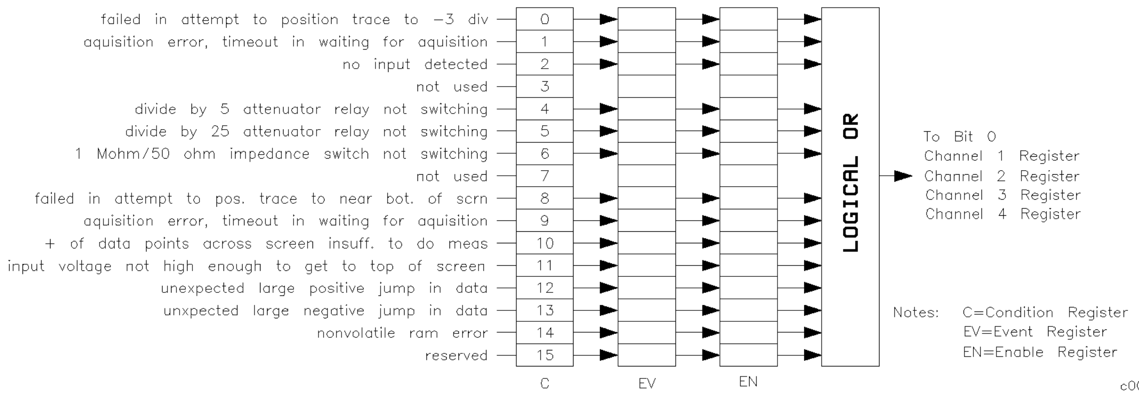

failed in attempt to position trace to −3 div
aquisition error, timeout in waiting for aquisition
no input detected
not used
divide by 5 attenuator relay not switching
divide by 25 attenuator relay not switching
1 Mohm/50 ohm impedance switch not switching
not used
failed in attempt to pos. trace to near bot. of scrn
aquisition error, timeout in waiting for aquisition
+ of data points across screen insuff. to do meas
input voltage not high enough to get to top of screen
unexpected large positive jump in data
unxpected large negative jump in data
nonvolatile ram error
reserved
0
1
2
3
4
5
6
7
8
9
10
11
12
13
14
15
C
EV
EN
LOGICAL OR
To Bit 0
Channel 1 Register
Channel 2 Register
Channel 3 Register
Channel 4 Register
Notes: C=Condition Register
EV=Event Register
EN=Enable Register
c005c

Figure 14-8. A/D Register

QUEStionable:CALibration:CHANnel<N>:DELay

The :SUMMARY:QUESTIONABLE:CALIBRATION:CHANNEL<N>:DELAY register reports the status of the delay calibration data for the specified channel (1 to 4). Use the following diagram to interpret the returned results. See figures 14-3, 14-4, and 14-5 for additional information on using the CONDITION?, ENABLE, ENABLE?, and [:EVENT]? commands.

Command Syntax

```
:SUMMary:QUEStionable:CALibration:CHANnel:{1|2|3|4}:DELay:ENABle
<number>
```

Example

```
:SUMM:QUES:CAL:CHAN3:DEL:ENAB 320
```

The example shown sets the enable mask, which allows true conditions (transitions) in the :CALIBRATION:CHANNEL3:DELAY:EVENT register to be reported. In this example, channel 1 calibration delay enable register bits 6 and 8 are set to true (1).

Query Syntax

```
:SUMMary:QUEStionable:CALibration:CHANnel{1|2|3|4}:DELay
{:CONDition?|:ENABle?|[:EVENt]?}
```

Example

```
:SUMM:QUES:CAL:CHAN2:DEL:COND?
```

The example shown queries the instrument to return the contents of the specified register.

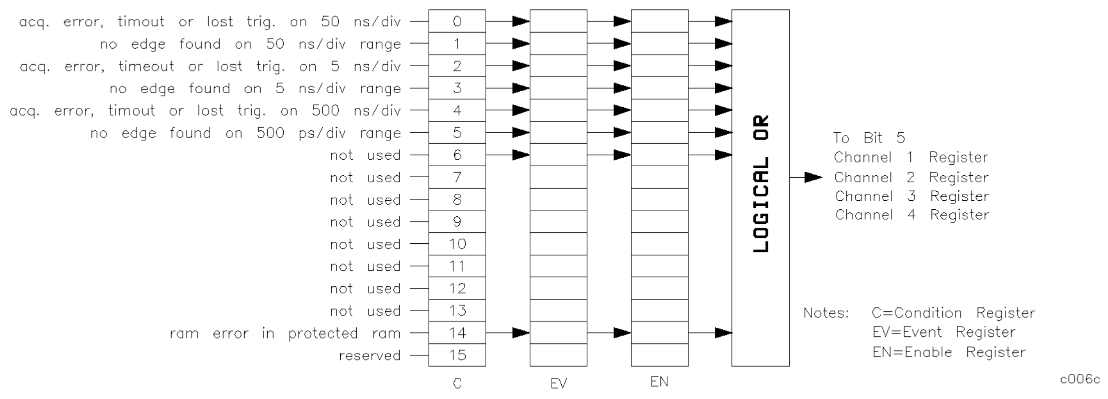

acq. error, timout or lost trig. on 50 ns/div
no edge found on 50 ns/div range
acq. error, timeout or lost trig. on 5 ns/div
no edge found on 5 ns/div range
acq. error, timout or lost trig. on 500 ns/div
no edge found on 500 ps/div range
not used
not used
not used
not used
not used
not used
not used
not used
ram error in protected ram
reserved
0
1
2
3
4
5
6
7
8
9
10
11
12
13
14
15
C
EV
EN
LOGICAL OR
To Bit 5
Channel 1 Register
Channel 2 Register
Channel 3 Register
Channel 4 Register
Notes: C=Condition Register
EV=Event Register
EN=Enable Register
c006c

Figure 14-9. Delay Register

QUEStionable:CALibration:CHANnel<N>:GAIN

The :SUMMARY:QUESTIONABLE:CALIBRATION:CHANNEL<N>:GAIN register reports the status of the gain calibration data for the specified channel (1 to 4). Use the following diagram to interpret the returned results. See figures 14-3, 14-4, and 14-5 for additional information on using the CONDITION?, ENABLE, ENABLE?, and [:EVENT]? commands.

Command Syntax

```
:SUMMary:QUEStionable:CALibration:CHANnel{1|2|3|4}:GAIN:ENABle
<number>
```

Example

```
:SUMM:QUES:CAL:CHAN2:GAIN:ENAB 16384
```

The example shown sets the enable mask, which allows true conditions (transitions) in the :CALIBRATION:CHANNEL2:GAIN:EVENT register to be reported. In this example, channel 2 calibration gain enable register bit 14 is set to true (1).

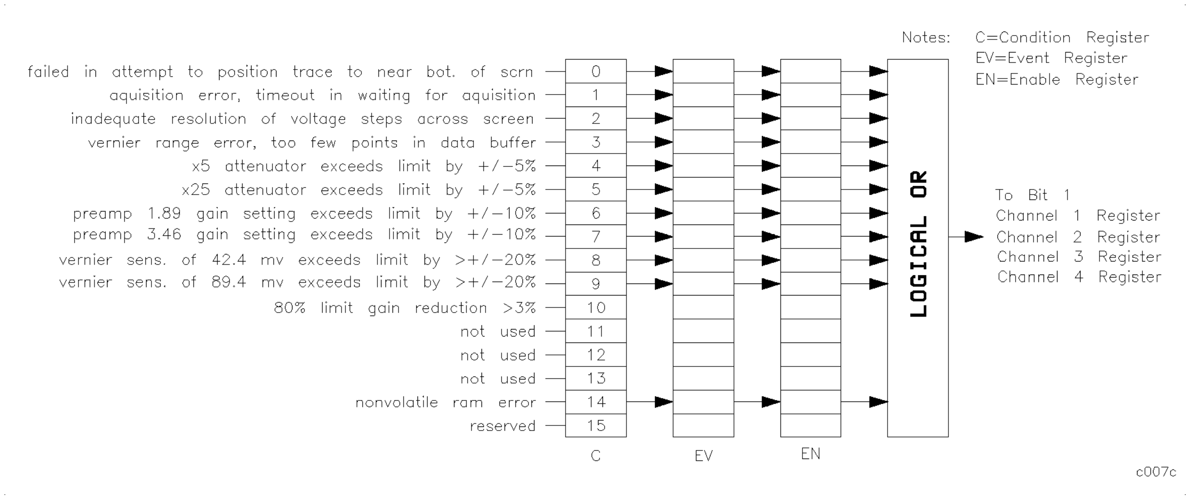
Query Syntax

```
:SUMMary:QUEStionable:CALibration:CHANnel{1|2|3|4}:GAIN
{:CONDition?|:ENABle?|[:EVENt]?}
```

Example

```
:SUMM:QUES:CAL:CHAN1:GAIN:ENAB?
```

The example shown queries the instrument to return the contents of the specified register.



Notes: C=Condition Register
EV=Event Register
EN=Enable Register
failed in attempt to position trace to near bot. of scrn
aquisition error, timeout in waiting for aquisition
inadequate resolution of voltage steps across screen
vernier range error, too few points in data buffer
x5 attenuator exceeds limit by +/-5%
x25 attenuator exceeds limit by +/-5%
preamp 1.89 gain setting exceeds limit by +/-10%
preamp 3.46 gain setting exceeds limit by +/-10%
vernier sens. of 42.4 mv exceeds limit by >+/-20%
vernier sens. of 89.4 mv exceeds limit by >+/-20%
80% limit gain reduction >3%
not used
not used
not used
nonvolatile ram error
reserved
0
1
2
3
4
5
6
7
8
9
10
11
12
13
14
15
C
EV
EN
LOGICAL OR
To Bit 1
Channel 1 Register
Channel 2 Register
Channel 3 Register
Channel 4 Register
c007c

Figure 14-10. Gain Register

QUEStionable:CALibration:CHANnel<N>:HYSTeresis

The :SUMMARY:QUESTIONABLE:CALIBRATION:CHANNEL<N>:HYSTeresis register reports the status of the hysteresis calibration data for the specified channel (1 to 4). Use the following diagram to interpret the returned results. See figures 14-3, 14-4, and 14-5 for additional information on using the CONDITION?, ENABLE, ENABLE?, and [:EVENT]? commands.

Command Syntax

```
:SUMMary:QUEStionable:CALibration:CHANnel {1|2|3|4}:HYSTeresis:
ENABle<number>
```

Example

```
:SUMM:QUES:CAL:CHAN1:HYST:ENAB 112
```

The example shown sets the enable mask, which allows true conditions (transitions) in the :CALIBRATION:CHANNEL1:HYSTERESIS:EVENT register to be reported. In this example, channel 1 calibration hysteresis enable register bits 4 through 6 are set to true (1).

Query Syntax

```
:SUMMary:QUEStionable:CALibration:CHANnel{1|2|3|4}:HYSTeresis
{:CONDition?|:ENABle?|[:EVENt]?}
```

Example

```
:SUMM:QUES:CAL:CHAN2:HYST:COND?
```

The example shown queries the instrument to return the contents of the specified register.

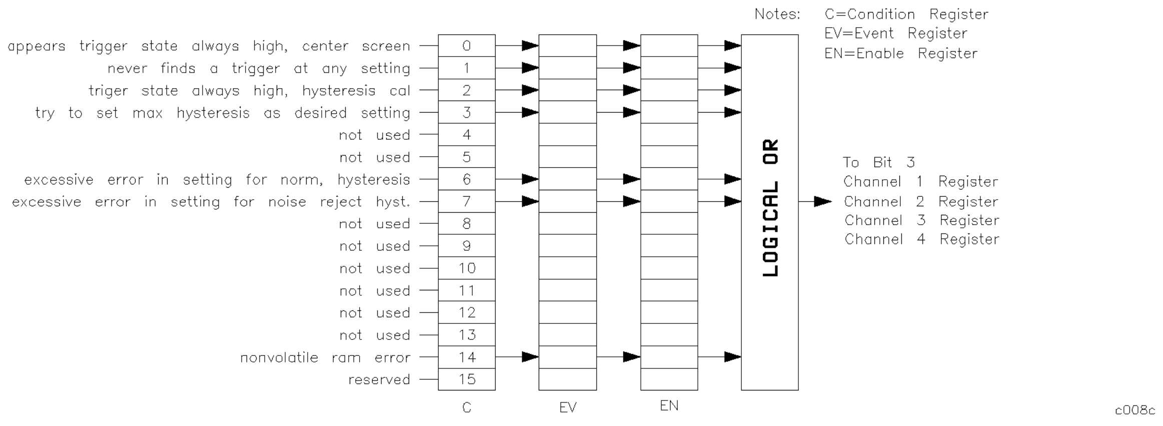

Notes: C=Condition Register
EV=Event Register
EN=Enable Register
appears trigger state always high, center screen
never finds a trigger at any setting
triger state always high, hysteresis cal
try to set max hysteresis as desired setting
not used
not used
excessive error in setting for norm, hysteresis
excessive error in setting for noise reject hyst.
not used
not used
not used
not used
not used
not used
nonvolatile ram error
reserved
0
1
2
3
4
5
6
7
8
9
10
11
12
13
14
15
LOGICAL OR
To Bit 3
Channel 1 Register
Channel 2 Register
Channel 3 Register
Channel 4 Register
C
EV
EN
c008c

Figure 14-11. Hysteresis Register

QUEStionable:CALibration:CHANnel<N>:OFFSet

The :SUMMARY:QUESTIONABLE:CALIBRATION:CHANNEL<N>:OFFSET register reports the status of the offset calibration data for the specified channel (1 to 4). Use the following diagram to interpret the returned results. See figures 14-3, 14-4, and 14-5 for additional information on using the CONDITION?, ENABLE, ENABLE?, and [:EVENT]? commands.

Command Syntax

```
:SUMMary:QUEStionable:CALibration:CHANnel {1|2|3|4}:OFFSet:ENABle
<number>
```

Example

```
:SUMM:QUES:CAL:CHAN4:OFFS:ENAB 11
```

The example shown sets the enable mask, which allows true conditions (transitions) in the :CALIBRATION:CHANNEL4:OFFSET:EVENT register to be reported. In this example, channel 4 calibration offset enable register bits 0, 1 and 3 are set to true (1).

Query Syntax

```
:SUMMary:QUEStionable:CALibration:CHANnel{1|2|3|4}:OFFSet
{:CONDition?|:ENABle?|[:EVENt]?}
```

Example

```
:SUMM:QUES:CAL:CHAN2:OFFS:COND?
```

The example shown queries the instrument to return the contents of the specified register.

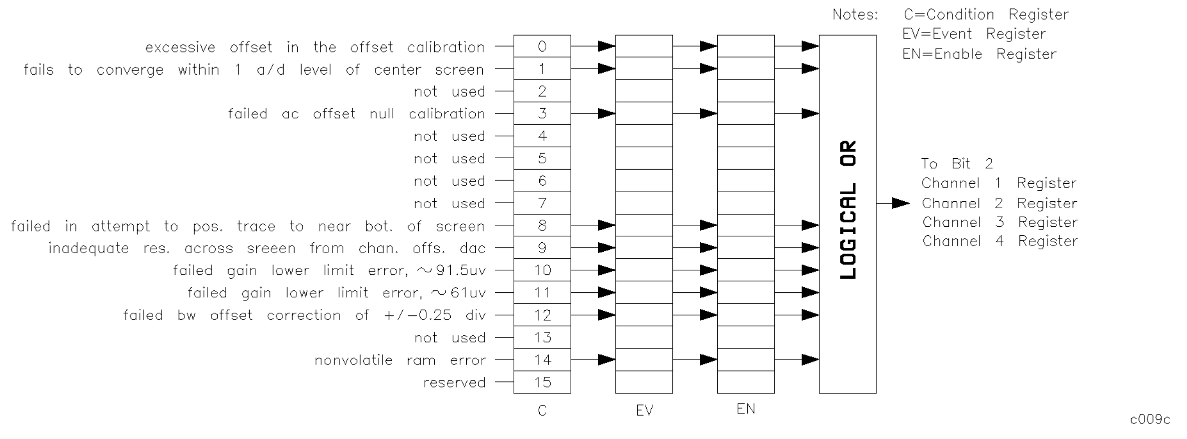

Notes: C=Condition Register
EV=Event Register
EN=Enable Register
excessive offset in the offset calibration
fails to converge within 1 a/d level of center screen
not used
failed ac offset null calibration
not used
not used
not used
not used
failed in attempt to pos. trace to near bot. of screen
inadequate res. across sreeen from chan. offs. dac
failed gain lower limit error, ~91.5uv
failed gain lower limit error, ~61uv
failed bw offset correction of +/-0.25 div
not used
nonvolatile ram error
reserved
0
1
2
3
4
5
6
7
8
9
10
11
12
13
14
15
C
EV
EN
LOGICAL OR
To Bit 2
Channel 1 Register
Channel 2 Register
Channel 3 Register
Channel 4 Register
c009c

Figure 14-12. Offset Register

QUEStionable:CALibration:CHANnel<N>:TNULl

The :SUMMARY:QUESTIONABLE:CALIBRATION:CHANNEL<N>:TNULL register reports the status of the time null calibration data for the specified channel (2 to 4). Channel 1 does not contain a time null register. Use the following diagram to interpret the returned results. See figures 14-3, 14-4, and 14-5 for additional information on using the CONDITION?, ENABLE, ENABLE?, and [:EVENT]? commands.

Command Syntax

```
:SUMMary:QUEStionable:CALibration:CHANnel{2|3|4}:TNULl:ENABle
<number>
```

Example

```
:SUMM:QUES:CAL:CHAN2:TNUL:ENAB 7680
```

The example shown sets the enable mask, which allows true conditions (transitions) in the :CALIBRATION:CHANNEL2:TNULL:EVENT register to be reported. In this example, channel 2 time null enable register bits 9 through 12 are set to true (1).

Query Syntax

```
:SUMMary:QUEStionable:CALibration:CHANnel{2|3|4}:TNULl
{:CONDition?|:ENABle?|[:EVENt]?}
```

Example

```
:SUMM:QUES:CAL:CHAN2:TNUL:ENAB?
```

The example shown queries the instrument to return the contents of the specified register.

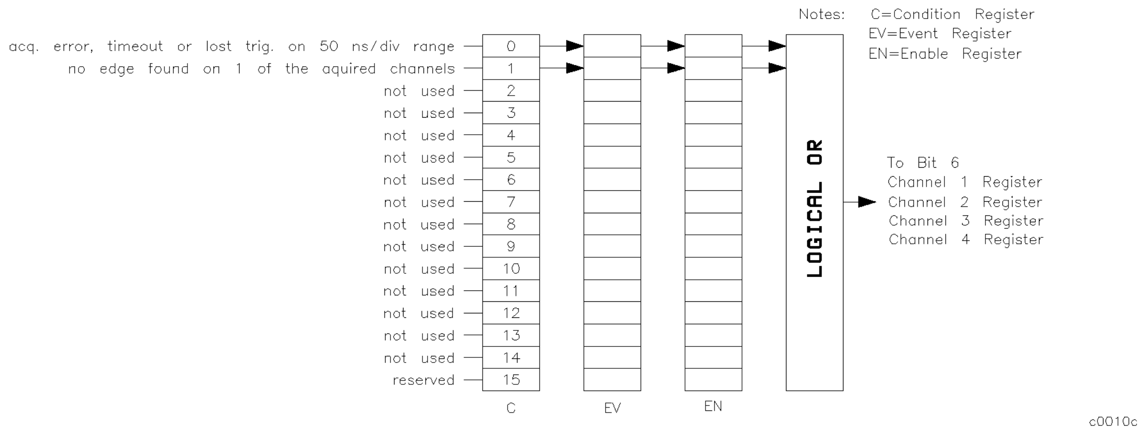


Figure 14-13. Time Null Register

QUEStionable:CALibration :CHANnel<N>:TRIGger

The :SUMMARY:QUESTIONABLE:CALIBRATION:CHANNEL<N>:TRIGGER register reports the status of the trigger calibration data for the specified channel (1 to 4). Use the following diagram to interpret the returned results. See figures 14-3, 14-4, and 14-5 for additional information on using the CONDITION?, ENABLE, ENABLE?, and [:EVENT]? commands.

Command Syntax

```
:SUMMary:QUEStionable:CALibration:CHANnel{1|2|3|4}:TRIGger:ENABle<number>
```

Example

```
:SUMM:QUES:CAL:CHAN2:TRIG:ENAB 112
```

The example shown sets the enable mask, which allows true conditions (transitions) in the :CALIBRATION:CHANNEL2:TRIGGER:EVENT register to be reported. In this example, channel 2 calibration trigger enable register bits 4 through 6 are set to true (1).

Query Syntax

```
:SUMMary:QUEStionable:CALibration:CHANnel{1|2|3|4}:TRIGger
{:CONDition?|:ENABle?|[:EVENt]?}
```

Example

```
:SUMM:QUES:CAL:CHAN2:TRIG:COND?
```

The example shown queries the instrument to return the contents of the specified register.

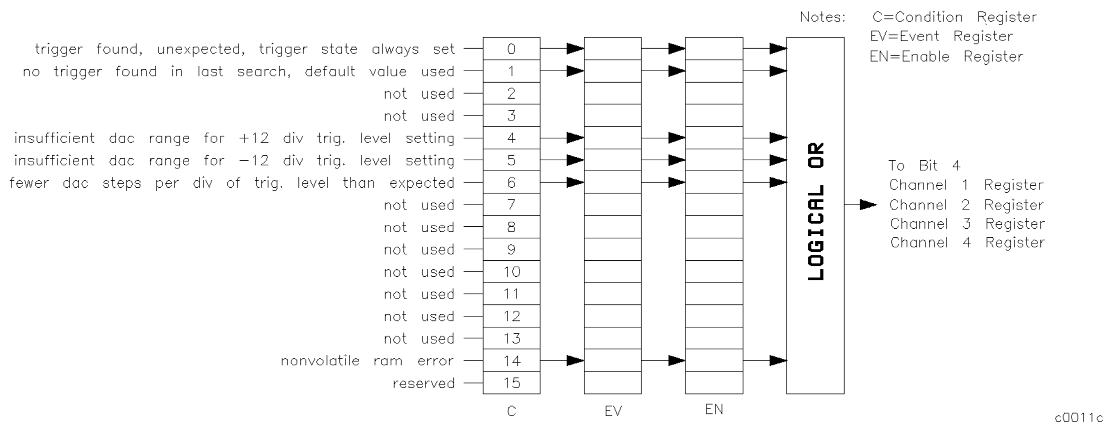

Notes: C=Condition Register
EV=Event Register
EN=Enable Register
trigger found, unexpected, trigger state always set
no trigger found in last search, default value used
not used
not used
insufficient dac range for +12 div trig. level setting
insufficient dac range for −12 div trig. level setting
fewer dac steps per div of trig. level than expected
not used
not used
not used
not used
not used
not used
not used
nonvolatile ram error
reserved
0
1
2
3
4
5
6
7
8
9
10
11
12
13
14
15
C
EV
EN
LOGICAL OR
To Bit 4
Channel 1 Register
Channel 2 Register
Channel 3 Register
Channel 4 Register
c0011c

Figure 14-14. Trigger Register

QUEStionable:CALibration:CHANnel1:LTRigger

The :SUMMARY:QUESTIONABLE:CALIBRATION:CHANNEL1:LTRIGGER register reports the status of logic trigger calibration data for channel 1. Only channel 1 contains the LTRIGGER register. Use the following diagram to interpret the returned results. See figures 14-3, 14-4, and 14-5 for additional information on using the CONDITION?, ENABLE, ENABLE?, and [:EVENT]? commands.

Command Syntax

```
:SUMMary:QUEStionable:CALibration:CHANnel1:LTRigger:ENABle
<number>
```

Example

```
:SUMM:QUES:CAL:CHAN1:LTR:ENAB 1008
```

The example shown sets the enable mask, which allows true conditions (transitions) in the :CALIBRATION:CHANNEL4:LTRIGGER:EVENT register to be reported. In this example, channel 1 calibration logic trigger enable register bits 4 through 9 are set to true (1).

Query Syntax

```
:SUMMary:QUEStionable:CALibration:CHANnel1:LTRigger
{:CONDition?|:ENABle?|[:EVENt]?}
```

Example

```
:SUMM:QUES:CAL:CHAN1:LTR:ENAB?
```

The example shown queries the instrument to return the contents of register 1.

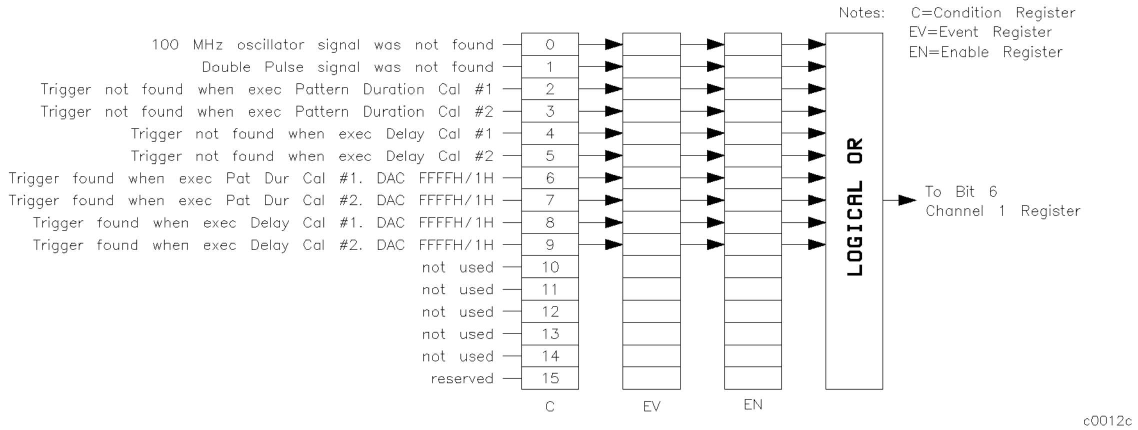

Notes: C=Condition Register
EV=Event Register
EN=Enable Register
100 MHz oscillator signal was not found 0
Double Pulse signal was not found 1
Trigger not found when exec Pattern Duration Cal #1 2
Trigger not found when exec Pattern Duration Cal #2 3
Trigger not found when exec Delay Cal #1 4
Trigger not found when exec Delay Cal #2 5
Trigger found when exec Pat Dur Cal #1. DAC FFFFH/1H 6
Trigger found when exec Pat Dur Cal #2. DAC FFFFH/1H 7
Trigger found when exec Delay Cal #1. DAC FFFFH/1H 8
Trigger found when exec Delay Cal #2. DAC FFFFH/1H 9
not used 10
not used 11
not used 12
not used 13
not used 14
reserved 15
C
EV
EN
LOGICAL OR
To Bit 6 Channel 1 Register
c0012c

Figure 14-15. Logic Trigger Register

QUEStionable:CALibration:DCALibration

The :SUMMARY:QUESTIONABLE:CALIBRATION:DCALIBRATION register reports the status of the default calibration factor data loaded. Use the following diagram to interpret the returned results. See figures 14-3, 14-4, and 14-5 for additional information on using the CONDITION?, ENABLE, ENABLE?, and [:EVENT]? commands.

Command Syntax

```
:SUMMary:QUEStionable:CALibration:DCALibration:ENABle <number>
```

Example

```
:SUMM:QUES:CAL:DCAL:ENAB 1
```

The example shown sets the enable mask, which allows true conditions (transitions) in the :CALIBRATION:DCALIBRATION:EVENT register to be reported. In this example, default calibration enable register bit 0 is set to true (1).

Query Syntax

```
:SUMMary:QUEStionable:CALibration:DCALibration{:CONDition?|
:ENABle?|[:EVENt]?}
```

Example

```
:SUMM:QUES:CAL::DCAL:ENAB?
```

The example shown queries the instrument to return the contents of the specified register.

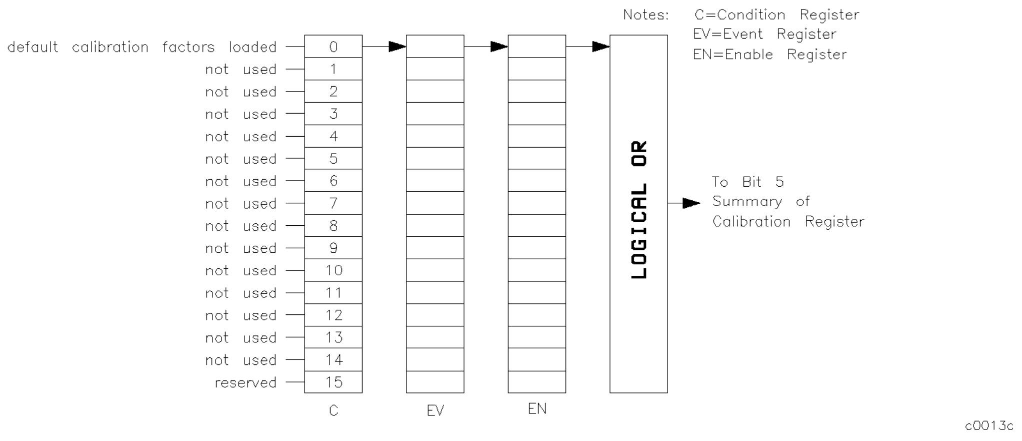

Notes: C=Condition Register
EV=Event Register
EN=Enable Register
default calibration factors loaded
not used
reserved
0
1
2
3
4
5
6
7
8
9
10
11
12
13
14
15
C
EV
EN
LOGICAL OR
To Bit 5 Summary of Calibration Register
c0013c

Figure 14-16. Default Cal Register

QUEStionable:CALibration:PROBe

The :SUMMARY:QUESTIONABLE:CALIBRATION:PROBE register reports probe calibration attenuation results. Use the following diagram to interpret the returned results. See figures 14-3, 14-4, and 14-5 for additional information on using the CONDITION?, ENABLE, ENABLE?, and [:EVENT]? commands.

Command Syntax

```
:SUMMary:QUEStionable:CALibration:PROBe:ENABle <number>
```

Example

```
:SUMM:QUES:CAL:PROB:ENAB 15
```

The example shown sets the enable mask, which allows true conditions (transitions) in the :CALIBRATION:PROBE:EVENT register to be reported. In this example, calibration probe attenuation enable register bits 0, through 3 are set to true (1).

Query Syntax

```
:SUMMary:QUEStionable:CALibration:PROBe{:CONDition?|:ENABle?|[:EVENt]?}
```

Example

```
:SUMM:QUES:CAL:PROB?
```

The example shown queries the instrument to return the contents of the specified register.

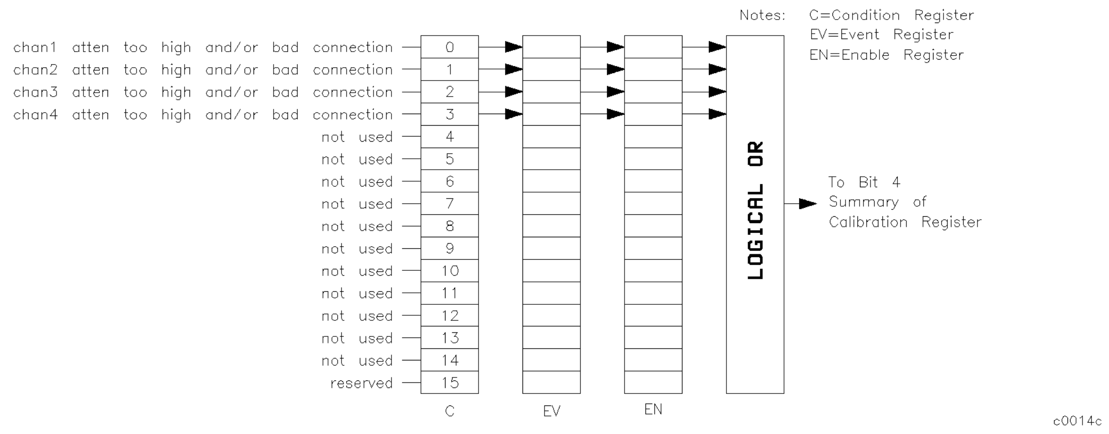

Notes: C=Condition Register
EV=Event Register
EN=Enable Register
chan1 atten too high and/or bad connection — 0
chan2 atten too high and/or bad connection — 1
chan3 atten too high and/or bad connection — 2
chan4 atten too high and/or bad connection — 3
not used — 4
not used — 5
not used — 6
not used — 7
not used — 8
not used — 9
not used — 10
not used — 11
not used — 12
not used — 13
not used — 14
reserved — 15
C
EV
EN
LOGICAL OR
To Bit 4 Summary of Calibration Register
c0014c

Figure 14-17. Probe Attenuation Register

QUEStionable:TEST

The :SUMMARY:QUESTIONABLE:TEST register reports diagnostic test results or self test status. Use the following diagram to interpret the returned results. See figures 14-3, 14-4, and 14-5 for additional information on using the CONDITION?, ENABLE, ENABLE?, and [:EVENT]? commands.

The assembly fail bits are only ever set after the :TEST:TALL command has been executed. If one of these bits is set it denotes that its associated assembly is probably faulty. These bits can be used as an aid to fault finding the instrument

Command Syntax

```
:SUMMary:QUEStionable:TEST:ENABle <number>
```

Example

```
:SUMM:QUES:TEST:ENAB 7
```

The example shown sets the enable mask, which allows true conditions (transitions) in the :TEST:EVENT register to be reported. In this example, test enable register bits 0 through 2 are set to true (1).

Query Syntax

```
:SUMMary:QUEStionable:TEST{:CONDition?|:ENABle?|[:EVENt]?}
```

Example

```
:SUMM:QUES:TEST:COND?
```

The example shown queries the instrument to return the contents of the specified register.

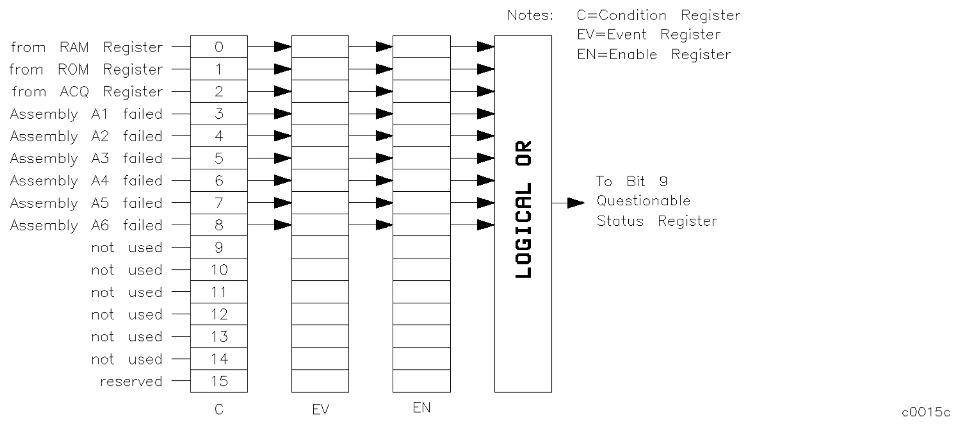


Figure 14-18. Self Test Status Register

QUEStionable:TEST:ACQuisition

The :SUMMARY:QUESTIONABLE:TEST:ACQUISITION register reports acquisition diagnostic test results. Use the following diagram to interpret the returned results. See figures 14-3, 14-4, and 14-5 for additional information on using the CONDITION?, ENABLE, ENABLE?, and [:EVENT]? commands.

Command Syntax

```
:SUMMary:QUEStionable:TEST:ACQuisition:ENABle <number>
```

Example

```
:SUMM:QUES:TEST:ACQ:ENAB 28
```

The example shown sets the enable mask, which allows true conditions (transitions) in the :TEST:ACQUISITION:EVENT register to be reported. In this example, test acquisition enable register bits 2 through 4 are set to true (1).

Query Syntax

```
:SUMMary:QUEStionable:TEST:ACQuisition{:CONDition?|:ENABle?|[:EVENt]?}
```

Example

```
:SUMM:QUES:TEST:ACQ:EVEN?
```

The example shown queries the instrument to return the contents of the specified register.

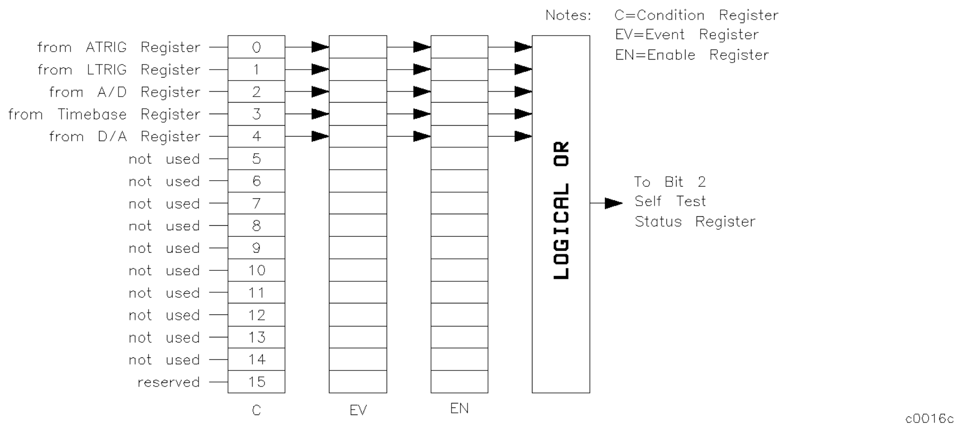


Figure 14-19. Acquisition Register

QUEStionable:TEST:ACQuisition:AD

The :SUMMARY:QUESTIONABLE:TEST:ACQUISITION:AD register reports acquisition A/D diagnostic test results. Use the following diagram to interpret the returned results. See figures 14-3, 14-4, and 14-5 for additional information on using the CONDITION?, ENABLE, ENABLE?, and [:EVENT]? commands.

Command Syntax

```
:SUMMary:QUEStionable:TEST:ACQuisition:AD:ENABle <number>
```

Example

```
:SUMM:QUES:TEST:ACQ:AD:ENAB 92
```

The example shown sets the enable mask, which allows true conditions (transitions) in the :TEST:ACQUISITION:AD:EVENT register to be reported. In this example, test acquisition A/D enable register bits 2 through 4 and bit 6 are set to true (1).

Query Syntax

```
:SUMMary:QUEStionable:TEST:ACQuisition:AD{:CONDition?|:ENABle?|[:EVENt]?}
```

Example

```
:SUMM:QUES:TEST:ACQ:AD?
```

The example shown queries the instrument to return the contents of the specified register.

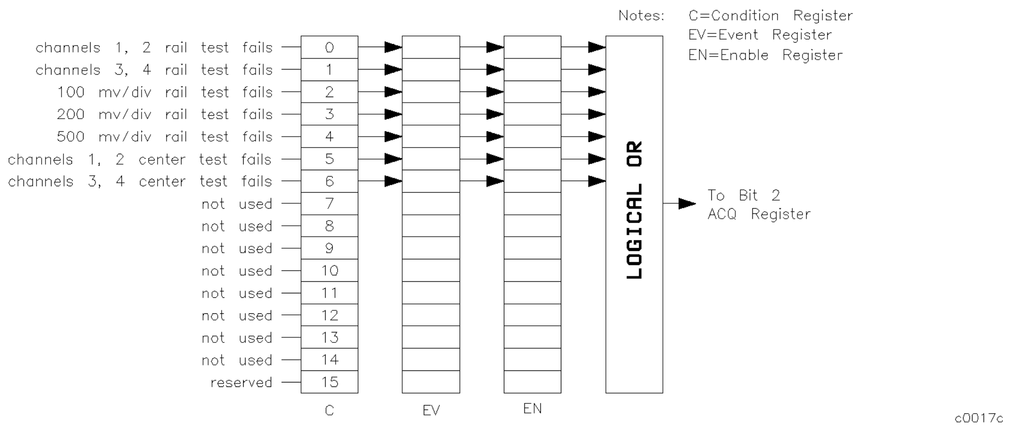

Notes: C=Condition Register
EV=Event Register
EN=Enable Register
channels 1, 2 rail test fails
channels 3, 4 rail test fails
100 mv/div rail test fails
200 mv/div rail test fails
500 mv/div rail test fails
channels 1, 2 center test fails
channels 3, 4 center test fails
not used
reserved
0
1
2
3
4
5
6
7
8
9
10
11
12
13
14
15
C
EV
EN
LOGICAL OR
To Bit 2 ACQ Register
c0017c

Figure 14-20. A/D Register

QUEStionable:TEST:ACQuisition:ATRigger

The :SUMMARY:QUESTIONABLE:TEST:ACQUISITION:ATRIGGER register reports acquisition analog trigger diagnostic test results. Use the following diagram to interpret the returned results. See figures 14-3, 14-4, and 14-5 for additional information on using the CONDITION?, ENABLE, ENABLE?, and [:EVENT]? commands.

Command Syntax

```
:SUMMary:QUEStionable:TEST:ACQuisition:ATRigger:ENABle <number>
```

Example

```
:SUMM:QUES:TEST:ACQ:ATR:ENAB 2
```

The example shown sets the enable mask, which allows true conditions (transitions) in the :TEST:ACQUISITION:ATRIGGER:EVENT register to be reported. In this example, test acquisition analog trigger enable register bit 1 is set to true (1).

Query Syntax

```
:SUMMary:QUEStionable:TEST:ACQuisition:ATRigger
{:CONDition?|:ENABle?|[:EVENt]?}
```

Example

```
:SUMM:QUES:TEST:ACQ:ATR:COND?
```

The example shown queries the instrument to return the contents of the specified register.

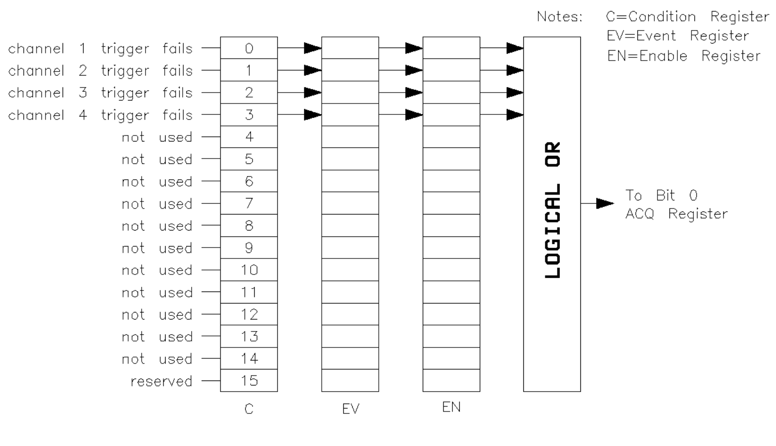


Figure 14-21. Analog Trigger Register

QUEStionable:TEST:ACQuisition:DA

The :SUMMARY:QUESTIONABLE:TEST:ACQUISITION:DA register reports acquisition D/A diagnostic test results. Use the following diagram to interpret the returned results. See figures 14-3, 14-4, and 14-5 for additional information on using the CONDITION?, ENABLE, ENABLE?, and [:EVENT]? commands.

Command Syntax

```
:SUMMary:QUEStionable:TEST:ACQuisition:DA:ENABle <number>
```

Example

```
:SUMM:QUES:TEST:ACQ:DA:ENAB 14
```

The example shown sets the enable mask, which allows true conditions (transitions) in the :TEST:ACQUISITION:DA:EVENT register to be reported. In this example, test acquisition D/A enable register bits 1 through 3 are set to true (1).

Query Syntax

```
:SUMMary:QUEStionable:TEST:ACQuisition:DA
{:CONDition?|:ENABle?|[:EVENt]?}
```

Example

```
:SUMM:QUES:TEST:ACQ:DA:ENAB?
```

The example shown queries the instrument to return the contents of the specified register.

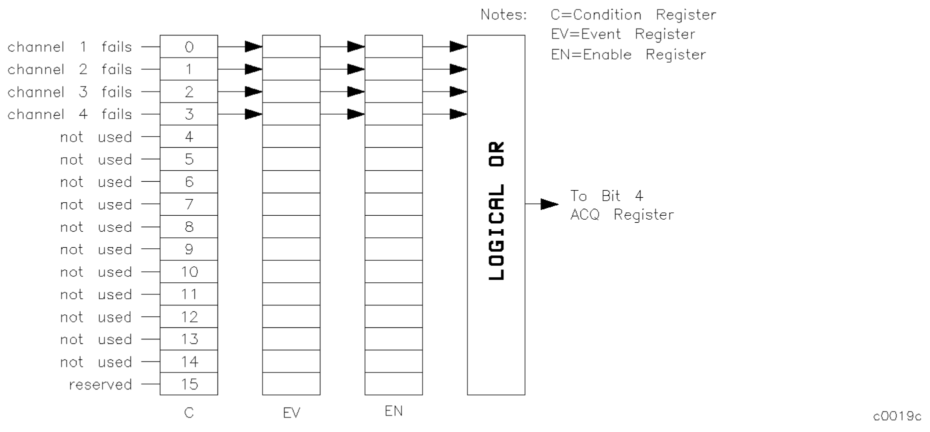


Figure 14-22. D/A Register

QUEStionable:TEST:ACQuisition:LTRigger

The :SUMMARY:QUESTIONABLE:TEST:ACQUISITION:LTRIGGER register reports acquisition logic trigger diagnostic test results. Use the following diagram to interpret the returned results. See figures 14-3, 14-4, and 14-5 for additional information on using the CONDITION?, ENABLE, ENABLE?, and [:EVENT]? commands.

Command Syntax

```
:SUMMary:QUEStionable:TEST:ACQuisition:LTRigger:ENABle <number>
```

Example

```
:SUMM:QUES:TEST:ACQ:LTR:ENAB 384
```

The example shown sets the enable mask, which allows true conditions (transitions) in the :TEST:ACQUISITION:LTRIGGER:EVENT Register to be reported. In this example, test acquisition analog trigger enable register bits 7 and 8 are set to true (1).

Query Syntax

```
:SUMMary:QUEStionable:TEST:ACQuisition:LTRigger
{:CONDition?|:ENABle?|[:EVENt]?}
```

Example

```
:SUMM:QUES:TEST:ACQ:LTR:EVEN?
```

The example shown queries the instrument to return the contents of the specified register.

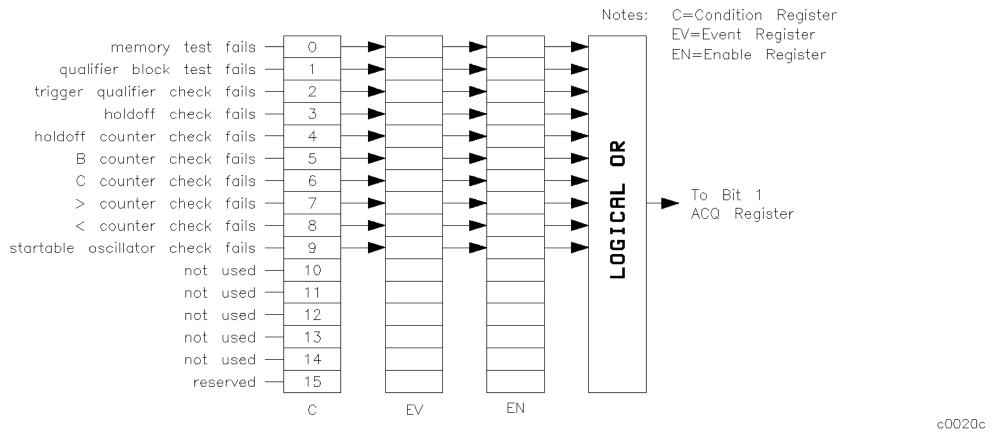


Figure 14-23. Logic Trigger Register

QUEStionable:TEST:ACQuisition:TIMebase

The :SUMMARY:QUESTIONABLE:TEST:ACQUISITION:TIMEBASE register reports acquisition logic trigger diagnostic test results. Use the following diagram to interpret the returned results. See figures 14-3, 14-4, and 14-5 for additional information on using the CONDITION?, ENABLE, ENABLE?, and [:EVENT]? commands.

Command Syntax

```
:SUMMary:QUEStionable:TEST:ACQuisition:TIMebase:ENABle <number>
```

Example

```
:SUMM:QUES:TEST:ACQ:TIM:ENAB 384
```

The example shown sets the enable mask, which allows true conditions (transitions) in the :TEST:ACQUISITION:TIMEBASE:EVENT register to be reported. In this example, test acquisition time base enable register bits 7 and 8 are set to true (1).

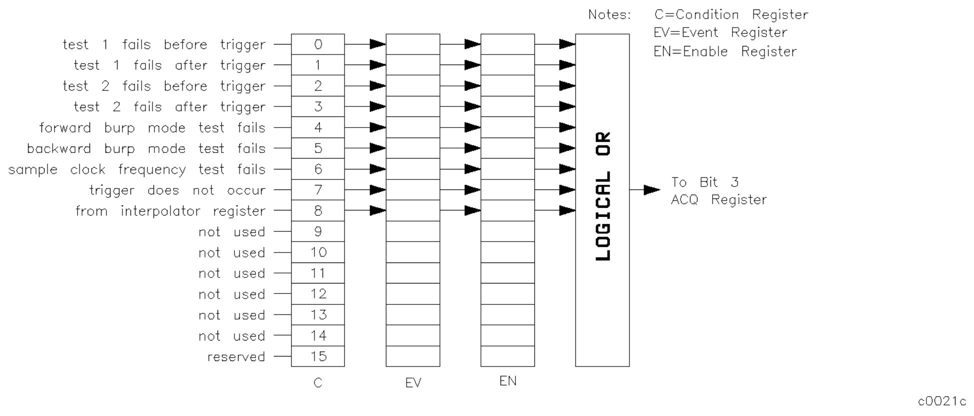
Query Syntax

```
:SUMMary:QUEStionable:TEST:ACQuisition:TIMebase
{:CONDition?|:ENABle?|[:EVENt]?}
```

Example

```
:SUMM:QUES:TEST:ACQ:TIM?
```

The example shown queries the instrument to return the contents of the specified register.



Notes: C=Condition Register
EV=Event Register
EN=Enable Register
test 1 fails before trigger
test 1 fails after trigger
test 2 fails before trigger
test 2 fails after trigger
forward burp mode test fails
backward burp mode test fails
sample clock frequency test fails
trigger does not occur
from interpolator register
not used
not used
not used
not used
not used
not used
reserved
0
1
2
3
4
5
6
7
8
9
10
11
12
13
14
15
C
EV
EN
LOGICAL OR
To Bit 3
ACQ Register
c0021c

Figure 14-24. Timebase Register

QUEStionable:TEST:ACQuisition:TIMebase:INTerpolator

The :SUMMARY:QUESTIONABLE:TEST:ACQUISITION:TIMEBASE:INTERPOLATOR register reports acquisition time base interpolator diagnostics. Use the following diagram to interpret the returned results. See figures 14-3, 14-4, and 14-5 for additional information on using the CONDITION?, ENABLE, ENABLE?, and [:EVENT]? commands.

Command Syntax

```
:SUMMary:QUEStionable:TEST:ACQuisition:TIMebase:INTerpolator:ENABle
<number>
```

Example

```
:SUMM:QUES:TEST:ACQ:TIM:INT:ENAB 12
```

The example shown sets the enable mask, which allows true conditions (transitions) in the :TEST:ACQUISITION:TIMEBASE:INTERPOLATOR:EVENT register to be reported. In this example, test acquisition time base enable register bits 2 and 3 are set to true (1).

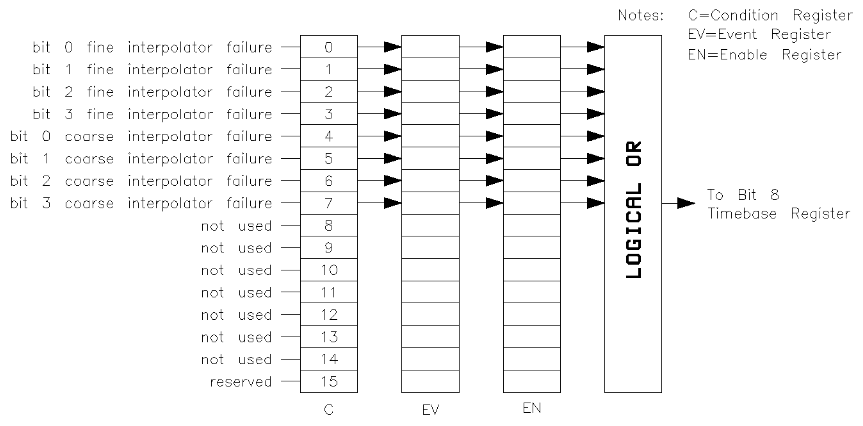
Query Syntax

```
:SUMMary:QUEStionable:TEST:ACQuisition:TIMebase:INTerpolator
{:CONDition?|:ENABle?|[:EVENt]?}
```

Example

```
:SUMM:QUES:TEST:ACQ:TIM:INT:COND?
```

The example shown queries the instrument to return the contents of the specified register.



Notes: C=Condition Register
EV=Event Register
EN=Enable Register
bit 0 fine interpolator failure
bit 1 fine interpolator failure
bit 2 fine interpolator failure
bit 3 fine interpolator failure
bit 0 coarse interpolator failure
bit 1 coarse interpolator failure
bit 2 coarse interpolator failure
bit 3 coarse interpolator failure
not used
not used
not used
not used
not used
not used
not used
reserved
0
1
2
3
4
5
6
7
8
9
10
11
12
13
14
15
LOGICAL OR
To Bit 8 Timebase Register
C
EV
EN

c0022c

Figure 14-25. Interpolator Register

QUEStionable:TEST:RAM

The :SUMMARY:QUESTIONABLE:TEST:RAM register reports random access memory diagnostic test results. Use the following diagram to interpret the returned results. See figures 14-3, 14-4, and 14-5 for additional information on using the CONDITION?, ENABLE, ENABLE?, and [:EVENT]? commands.

Command Syntax

```
:SUMMary:QUEStionable:TEST:RAM:ENABle <number>
```

Example

```
:SUMM:QUES:TEST:RAM:ENAB 12
```

The example shown sets the enable mask, which allows true conditions (transitions) in the :TEST:RAM:EVENT register to be reported. In this example, test acquisition time base enable register bits 2 and 3 are set to true (1).

Query Syntax

```
:SUMMary:QUEStionable:TEST:RAM{:CONDition?|:ENABle?|[:EVENt]?}
```

Example

```
:SUMM:QUES:TEST:RAM:COND?
```

The example shown queries the instrument to return the contents of the specified register.

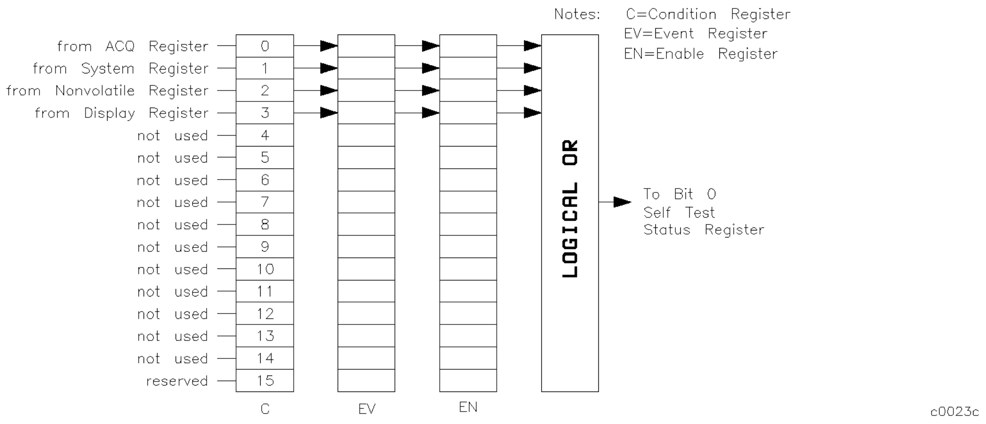


Figure 14-26. RAM Register

QUEStionable:TEST:RAM:ACQuisition

The :SUMMARY:QUESTIONABLE:TEST:RAM:ACQUISITION register reports acquisition random access memory diagnostic test results. Use the following diagram to interpret the returned results. See figures 14-3, 14-4, and 14-5 for additional information on using the CONDITION?, ENABLE, ENABLE?, and [:EVENT]? commands.

Command Syntax

```
:SUMMary:QUEStionable:TEST:RAM:ACQuisition:ENABle <number>
```

Example

```
:SUMM:QUES:TEST:RAM:ACQ:ENAB 3
```

The example shown sets the enable mask, which allows true conditions (transitions) in the :TEST:RAM:ACQUISITION:EVENT register to be reported. In this example, test acquisition time base enable register bits 0 and 1 are set to true (1).

Query Syntax

```
:SUMMary:QUEStionable:TEST:RAM:ACQuisition{:CONDition?|:ENABle?|[:EVENt]?}
```

Example

```
:SUMM:QUES:TEST:RAM:ACQ?
```

The example shown queries the instrument to return the contents of the specified register.

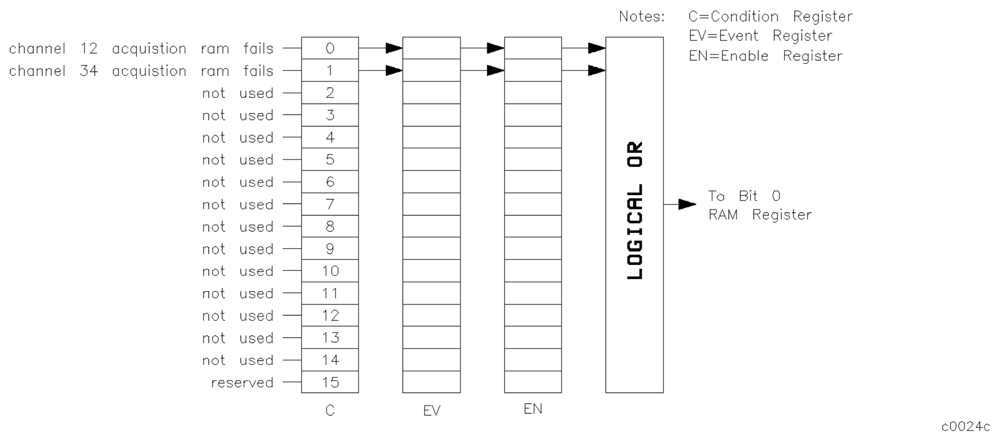


Figure 14-27. Acquisition Register

QUEStionable:TEST:RAM:DISPlay

The :SUMMARY:QUESTIONABLE:TEST:RAM:DISPLAY register reports display random access memory test results. Use the following diagram to interpret the returned results. See figures 14-3, 14-4, and 14-5 for additional information on using the CONDITION?, ENABLE, ENABLE?, and [:EVENT]? commands.

Note In the HP 70703A the display memory is **NOT** the memory used by the MMS graphics device. Failures reported by the register should not affect correct instrument operation.

Command Syntax

```
:SUMMary:QUEStionable:TEST:RAM:DISPlay:ENABle <number>
```

Example

```
:SUMM:QUES:TEST:RAM:DISP:ENAB 3
```

The example shown sets the enable mask, which allows true conditions (transitions) in the :TEST:RAM:DISPLAY:EVENT register to be reported. In this example, test acquisition time base enable register bits 0 and 1 are set to true (1).

Query Syntax

```
:SUMMary:QUEStionable:TEST:RAM:DISPlay{:CONDition?|:ENABle?|[:EVENt]?}
```

Example

```
:SUMM:QUES:TEST:RAM:DISP?
```

The example shown queries the instrument to return the contents of the specified register.

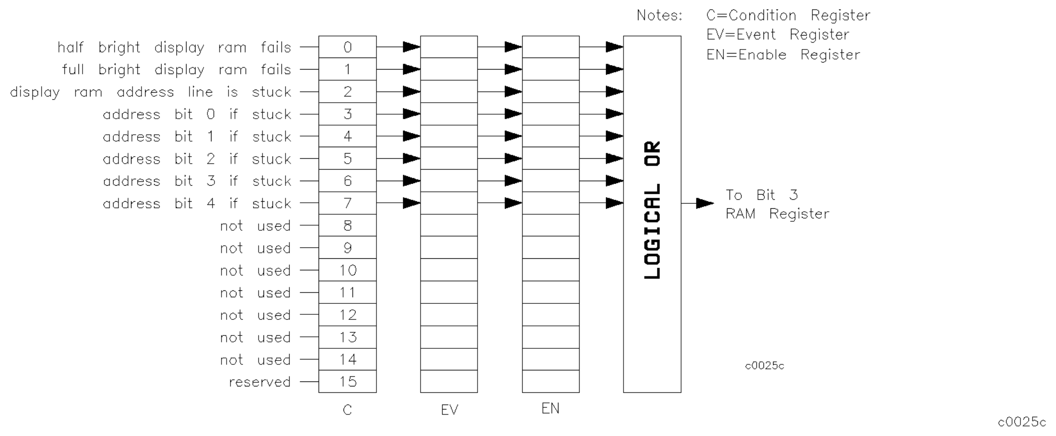


Figure 14-28. Display Register

QUEStionable:TEST:RAM:NVOLatile

The :SUMMARY:QUESTIONABLE:TEST:RAM:NVOLATILE register reports nonvolatile random access memory diagnostic test results. Use the following diagram to interpret the returned results. See figures 14-3, 14-4, and 14-5 for additional information on using the CONDITION?, ENABLE, ENABLE?, and [:EVENT]? commands.

Command Syntax

```
:SUMMary:QUEStionable:TEST:RAM:NVOLatile:ENABle <number>
```

Example

```
:SUMM:QUES:TEST:RAM:NVOL:ENAB 24
```

The example shown sets the enable mask, which allows true conditions (transitions) in the :TEST:RAM:NVOLATILE:EVENT register to be reported. In this example, test acquisition time base enable register bits 3 and 4 are set to true (1).

Query Syntax

```
:SUMMary:QUEStionable:TEST:RAM:NVOLatile{:CONDition?|:ENABle?|[:EVENt]?}
```

Example

```
:SUMM:QUES:TEST:RAM:NVOL:COND?
```

The example shown queries the instrument to return the contents of the specified register.

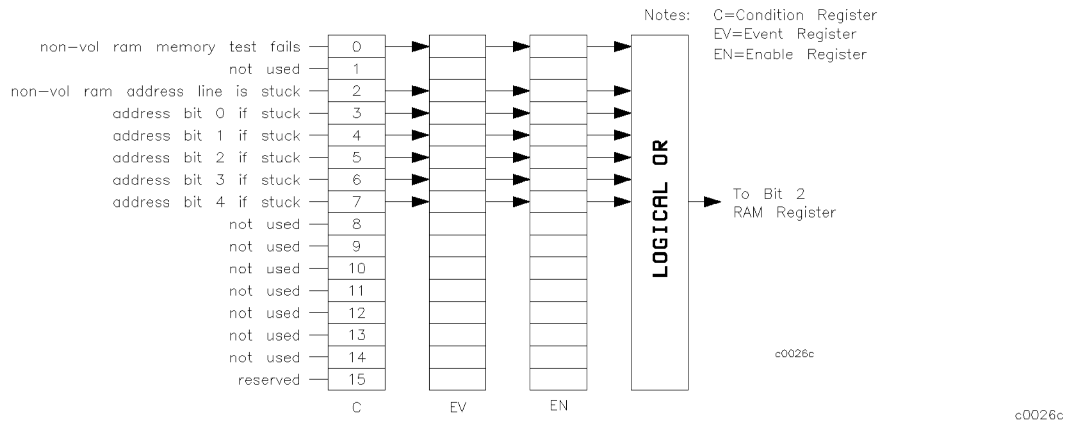


Figure 14-29. Nonvolatile Register

QUEStionable:TEST:RAM:SYSTem

The :SUMMARY:QUESTIONABLE:TEST:RAM:SYSTEM register reports system random access memory diagnostic test results. Use the following diagram to interpret the returned results. See figures 14-3, 14-4, and 14-5 for additional information on using the CONDITION?, ENABLE, ENABLE?, and [:EVENT]? commands.

Command Syntax

```
:SUMMary:QUEStionable:TEST:RAM:SYSTem:ENABle <number>
```

Example

```
:SUMM:QUES:TEST:RAM:SYST:ENAB 24
```

The example shown sets the enable mask, which allows true conditions (transitions) in the :TEST:RAM:SYSTEM:EVENT register to be reported. In this example, test acquisition time base enable register bits 3 and 4 are set to true (1).

Query Syntax

```
:SUMMary:QUEStionable:TEST:RAM:SYSTem{:CONDition?|:ENABle?|[:EVENt]?}
```

Example

```
:SUMM:QUES:TEST:RAM:SYST?
```

The example shown queries the instrument to return the contents of the specified register.

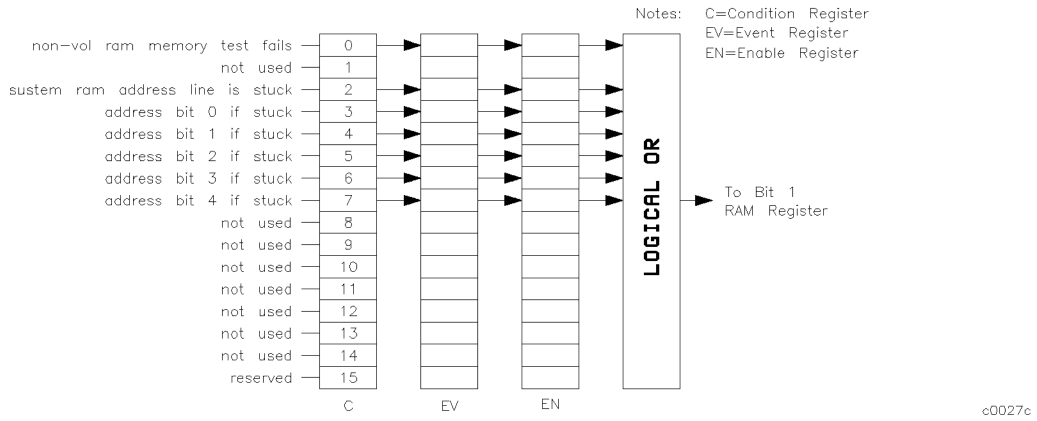


Figure 14-30. System Register

QUEStionable:TEST:ROM:NPRotect

The :SUMMary:QUEStionable:TEST:ROM register reports read only memory diagnostic test results. Use the following diagram to interpret the returned results. See figures 14-3, 14-4, and 14-5 for additional information on using the CONDITION?, ENABLE, ENABLE?, and [:EVENT]? commands.

Command Syntax

```
:SUMMary:QUEStionable:TEST:ROM:ENABle <number>
```

Example

```
:SUMM:QUES:TEST:ROM:ENAB 3
```

The example shown sets the enable mask, which allows true conditions (transitions) in the :TEST:ROM:EVENT register to be reported. In this example, test acquisition time base enable register bits 0 and 1 are set to true (1).

Query Syntax

```
:SUMMary:QUEStionable:TEST:ROM{:CONDition?|:ENABle?|[:EVENt]?}
```

Example

```
:SUMM:QUES:TEST:ROM:ENAB?
```

The example shown queries the instrument to return the contents of the specified register.

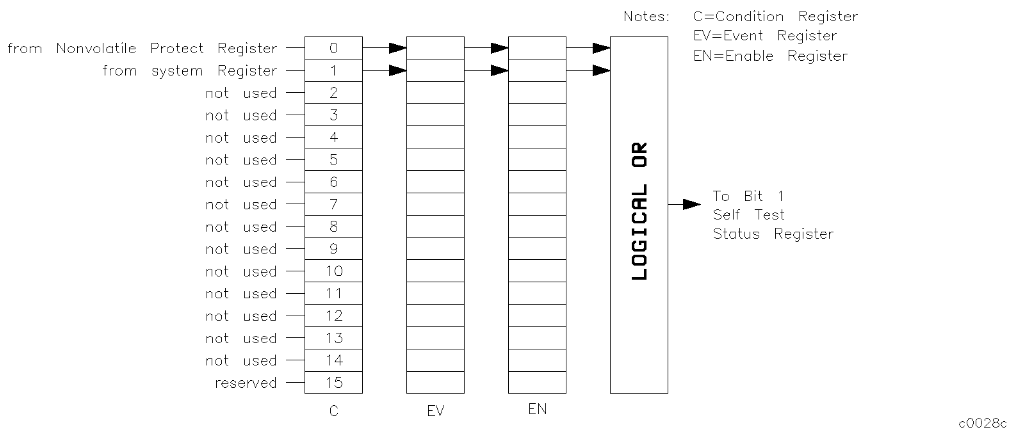


Figure 14-31. ROM Register

QUEStionable:TEST:ROM:NPRotect

The :SUMMARY:QUESTIONABLE:TEST:ROM:NPROTECT register reports nonvolatile protected random access memory diagnostic test results. Use the following diagram to interpret the returned results. See figures 14-3, 14-4, and 14-5 for additional information on using the CONDITION?, ENABLE, ENABLE?, and [:EVENT]? commands.

Command Syntax

```
:SUMMary:QUEStionable:TEST:ROM:NPRotect <number>
```

Example

```
:SUMM:QUES:TEST:ROM:NPR 1
```

The example shown sets the enable mask, which allows true conditions (transitions) in the :TEST:ROM:NPROTECT:EVENT register to be reported. In this example, test acquisition time base enable register bit 0 is set to true (1).

Query Syntax

```
:SUMMary:QUEStionable:TEST:ROM:NPRotect{:CONDition?|:ENABle?|[:EVENt]?}
```

Example

```
:SUMM:QUES:TEST:ROM:NPR:EVEN?
```

The example shown queries the instrument to return the contents of the specified register.

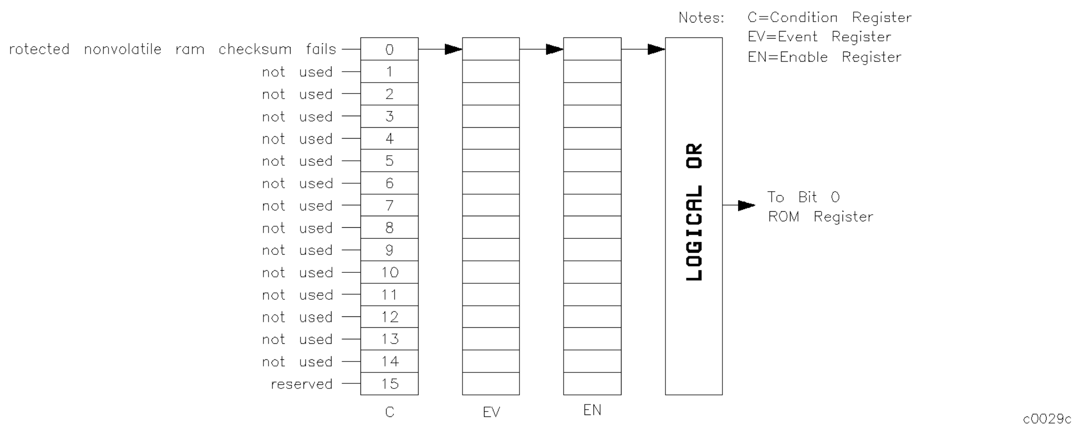


Figure 14-32. Nonvolatile Protect Register

QUEStionable:TEST:ROM:SYSTem

The :SUMMARY:QUESTIONABLE:TEST:ROM:SYSTEM register reports system read only memory diagnostic test results. Use the following diagram to interpret the returned results. See figures 14-3, 14-4, and 14-5 for additional information on using the CONDITION?, ENABLE, ENABLE?, and [:EVENT]? commands.

Command Syntax

```
:SUMMary:QUEStionable:TEST:ROM:SYSTem <number>
```

Example

```
:SUMM:QUES:TEST:ROM:SYST 7
```

The example shown sets the enable mask, which allows true conditions (transitions) in the :TEST:ROM:SYSTEM register to be reported. In this example, test acquisition time base enable register bits 0 through 2 are set to true (1).

Query Syntax

```
:SUMMary:QUEStionable:TEST:ROM:SYSTem{:CONDition?|:ENABle?|[:EVENt]?}
```

Example

```
:SUMM:QUES:TEST:ROM:SYST:ENAB?
```

The example shown queries the instrument to return the contents of the specified register.

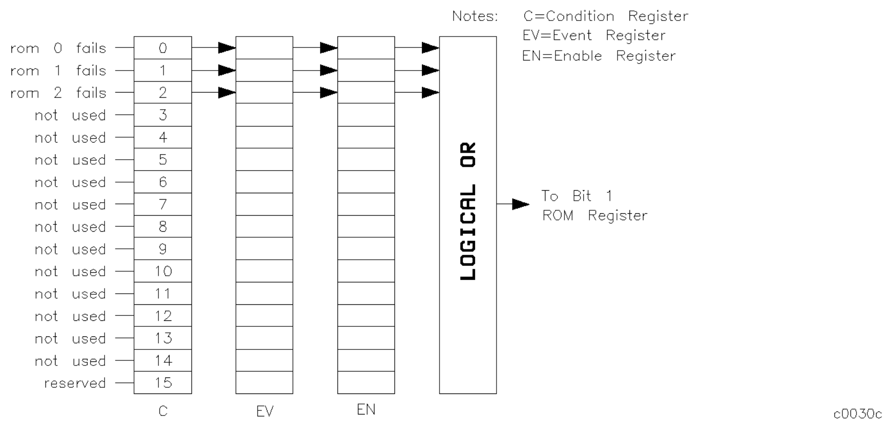


Figure 14-33. ROM System Register

QUEStionable:TIME

The :SUMMARY:QUESTIONABLE:TIME register reports time measurement status. Use the following diagram to interpret the returned results. See figures 14-3, 14-4, and 14-5 for additional information on using the CONDITION?, ENABLE, ENABLE?, AND [:EVENT]? commands.

Command Syntax

```
:SUMMary:QUEStionable:TIME<number>
```

Example

```
:SUMM:QUES:TIME 1
```

The example shown sets the enable mask, which allows true conditions (transistions) in the :TIME:EVENT register to be reported. In this example, time enable register bit 0 is set to true (1).

Query Syntax

```
:SUMMary:QUEStionable:TIME{:CONDition?|:ENABle?|[:EVENt]?}
```

Example

```
:SUMM:QUES:TIME:EVEN?
```

The example shown queries the instrument to return the contents of the specified register.

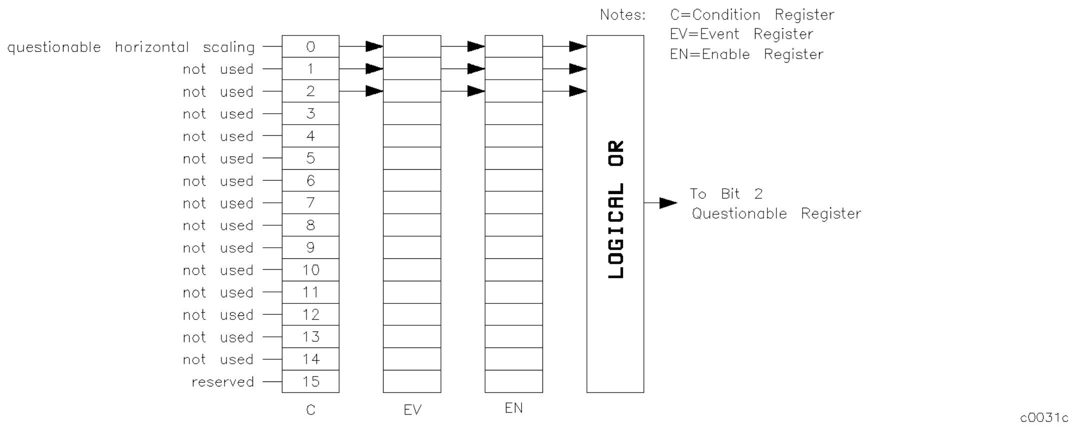


Figure 14-34. Time Register

15

Test Subsystem

The TEST subsystem is used to perform internal diagnostics. These diagnostics are provided to give a high confidence level of instrument functionality. Before performing any of the diagnostics, execute a *RST to set critical parameters to a known state, and a :SUMMARY:PRESET to enable the Summary Questionable registers.

The results of the test are stored in the :SUMMARY:QUESTIONABLE:TEST register subsystem.

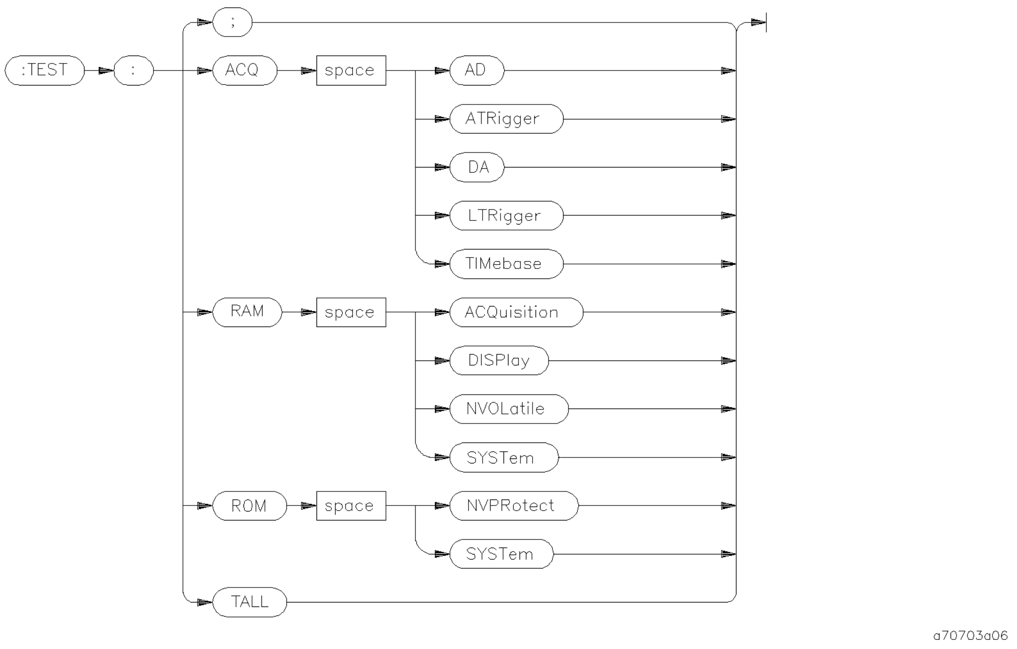


Figure 15-1. TEST Subsystem Commands Syntax Diagram

ACQ

The :TEST:ACQ command is used to perform up to five acquisition tests. When selected the HP 70703A performs an Analog Trigger test, Logic Trigger test, an A/D test, a Time base test, and/or a D/A test. If the <test> parameter is not sent, all five tests are performed.

Command Syntax

```
:TEST:ACQ <test>
```

Where:

```
<test> ::={AD|ATRigger|DA|LTRigger|TIMebase}
```

Example

```
OUTPUT 707;":TEST:ACQ TIM"
```

RAM

The :TEST:RAM command is used to perform up to four random access memory tests. When selected the HP 70703A performs a Display RAM test, System RAM test, a Nonvolatile RAM test, and/or an Acquisition RAM test. If the <test> parameter is not sent, all four tests are performed.

Command Syntax

```
:TEST:RAM <test>


<test> ::={ACQuisition|DISPlay|NVOLatile|SYSTem}
```

Example

```
OUTPUT 707;":TEST:RAM"
```

Note The example shown will perform all four RAM tests.

Note In the HP 70703A, the display memory is not the memory used by the MMS graphics device.

ROM

The :TEST:ROM command is used to perform one read-only memory test and one nonvolatile protected random-access memory test. When selected, the HP 70703A performs a System ROM test and/or a Protected Nonvolatile RAM test. If the <test> parameter is not sent, both tests are performed.

Command Syntax

```
:TEST:ROM <test>

<text> ::={NVPRotect|SYSTem}
```

Example

```
OUTPUT 707;":TEST:ROM NVPR"
```

TALL

The :TEST:TALL command is used to perform the RAM, ROM, and Acquisition tests. When selected, the HP 70703A performs all the individual tests.

This command will set the assembly fail bits in the :SUMMARY:QUESTIONABLE:TEST register, in addition to the other test fail and summary bits. Refer to Chapter 14 for more information.

Command Syntax

```
:TEST:TALL
```

Example

```
OUTPUT 707;":TEST:TALL"
```

Timebase Subsystem

The TIMEBASE subsystem controls the horizontal axis, “X axis,” oscilloscope functions.

The TIMEBASE subsystem also contains the commands that control the Timebase Window mode. The Timebase Window mode allows a second timebase to be used. The window settings are :WINDOW:DELAY (window position) and :WINDOW:RANGE (window width).

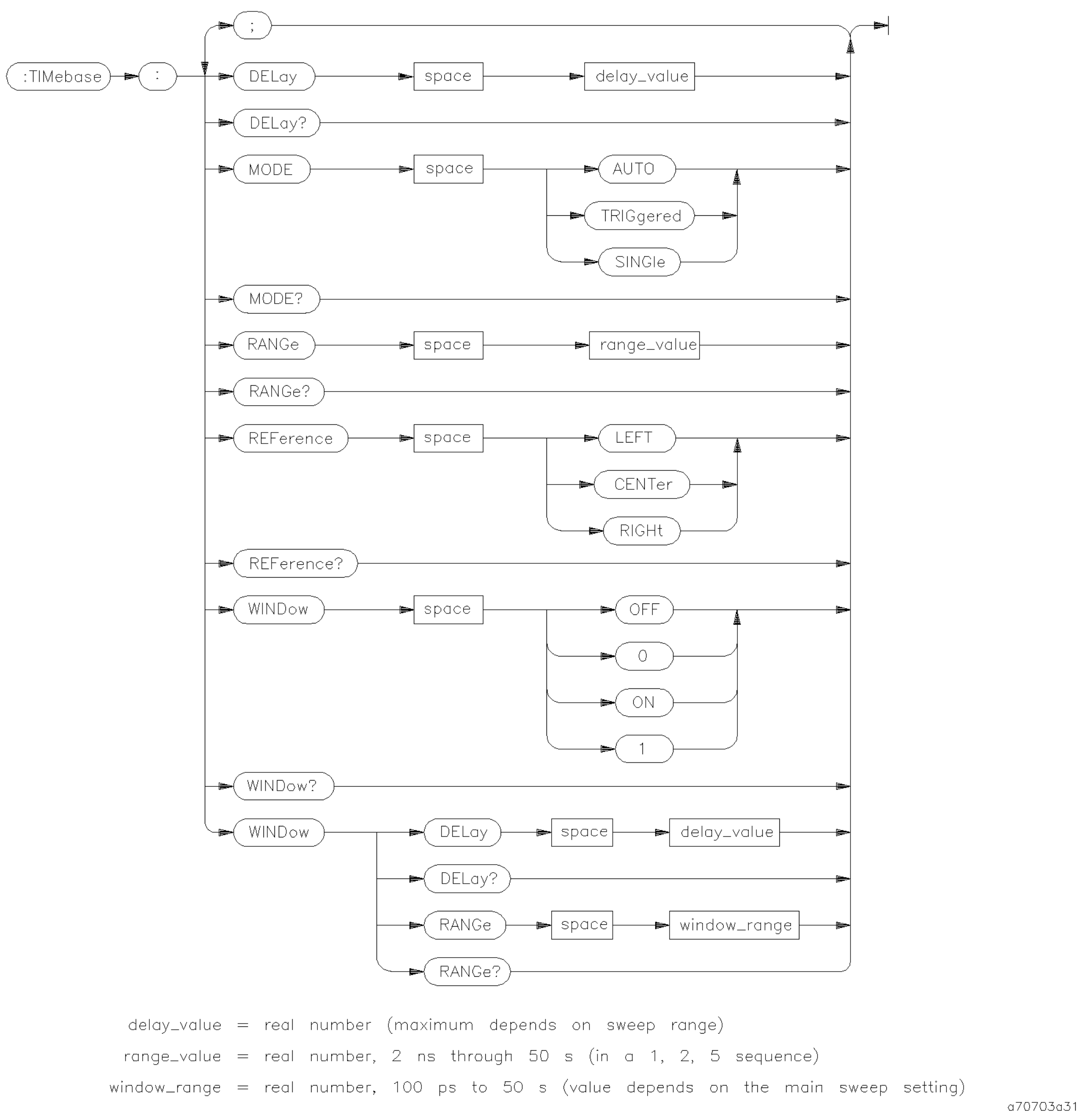


Figure 16-1. TIMEBASE Subsystem Commands Syntax Diagram

DELay

The :TIMEBASE:DELAY command sets the timebase delay. This delay is the time interval between the trigger event and the active waveform delay reference point. The delay reference point is set to the left, right, or center of the active waveform using the :TIMEBASE:REFERENCE command.

The DELAY query returns the current delay value.

Command Syntax

```
:TIMebase:DELay <delay>
```

Where:

```
<delay> ::= time in seconds from trigger to on screen delay reference
            point. Maximum value depends on time/division setting.
```

Example

```
OUTPUT 707;":TIM:DEL 2E-3"
```

Query Syntax

```
:TIMebase:DELay?
```

Returned Format

```
<delay><NL>
```

Where:

```
<delay> ::= time from trigger to display reference in seconds
            Display reference is left, center, or right
            (exponential - NR3 format)
```

Example

```
DIM D1$[50]
OUTPUT 707;":TIMEBASE:DELAY?"
ENTER 707;D1$
PRINT D1$
```

MODE

The :TIMEBASE:MODE command selects the timebase mode. This function defines when data will be acquired with respect to triggering.

If the AUTO mode is selected, the unit will provide a baseline in the absence of a signal and will acquire data regardless of trigger requirements. If a signal is present but the oscilloscope is not triggered, the waveform will be unsynchronized (not be a baseline) and available data will be acquired.

If the TRIGGERED mode is selected and no trigger is present, the unit will not sweep, and the data acquired on the previous trigger will remain.

If the SINGLE mode is selected, the present waveform will be cleared and the HP 70703A will stop acquiring data. The RUN command will arm the trigger, then data will be acquired when the trigger is found. To make a single acquisition a RUN command should be sent.

The MODE query returns the current mode.

Command Syntax

```
:TIMebase:MODE {AUTO|TRIGgered|SINGle}
```

Example

```
OUTPUT 707;":TIM:MODE TRIGGERED"
```

Query Syntax

```
:TIMebase:MODE?
```

Returned Format

```
<mode><NL>

<mode> ::= {AUTO|TRIGgered|SINGle}
```

Example

```
DIM Mode$[30]
OUTPUT 707;"TIMEBASE:MODE?"
ENTER 707;Mode$
PRINT Mode$
```

RANGe

The :TIMEBASE:RANGE command sets the full-scale horizontal time in seconds. The RANGE value is ten times the time per division.

The RANGE query returns the current range value.

Command Syntax

```
:TIMebase:RANGe <range>

<range> ::= 2 ns to 50 s in a 1,2,5 sequence"
```

Example

```
OUTPUT 707;":TIM:RANG 0.1"
```

Query Syntax

```
:TIMebase:RANGe?
```

Returned Format

```
<range><NL>
```

Where:

```
<range> ::= 2 ns to 50 s (exponential - NR3 format)
```

Example

```
DIM Rng$[50]
OUTPUT 707;":TIMEBASE:RANGE?"
ENTER 707;Rng$
PRINT Rng$
```

Note Changes in the range parameters may affect the current settings specified for :TIMEBASE:DELAY, and :TIMEBASE:WINDOW:RANGE.

REFerence

The :TIMEBASE:REFERENCE command sets the display reference to the left, right, or to the center of the active waveform.

The REFERENCE query returns the current display reference.

Command Syntax

```
:TIMebase:REFerence {LEFT|CENTer|RIGHt}
```

Example

```
OUTPUT 707;":TIMEBASE:REFERENCE LEFT"
```

Query Syntax

```
:TIMebase:REFerence?
```

Returned Format

```
{LEFT|CENTer|RIGHt}<NL>
```

Example

```
DIM Rf$[30]
OUTPUT 707;":TIMEBASE:REFERENCE?"
ENTER 707;Rf$
PRINT Rf$
```

WINDow

The :TIMEBASE:WINDOW command controls whether or not the second (expanded) timebase is in use. If this command is set to ON, the expanded timebase is part of the active waveform and all measurements are taken on the data present in the second (expanded) timebase.

When the timebase window is on, the expanded timebase data can be acquired by sending the WAVEFORM:DATA? query. However care must be taken in order to ensure valid data is present when the data is requested.

The WINDOW query returns the current state of this command.

Command Syntax

```
:TIMebase:WINDow {{ON|1}|{OFF|0}}
```

Example

```
OUTPUT 707;":TIM:WIND 1"
```

Query Syntax

```
:TIMbase:WINDow?
```

Returned Format

```
{1|0}<NL>
```

Example

```
DIM Tw$[50]
OUTPUT 707;":TIMEBASE:WINDOW?"
ENTER 707;"Tw$
PRINT Tw$
```

WINDow:DELay (Position)

The :TIMEBASE:WINDOW:DELAY command sets the timebase window delay. The window delay actually sets the position of the expanded timebase window on the main sweep. The range for this command is determined by the main sweep seconds/division and delay values. The value for this command must keep the window on the main sweep.

The WINDOW:DELAY query returns the current value.

Command Syntax

```
:TIMebase:WINDow:DELay <wid_delay>
```

Where:

```
<wid_delay> ::= time in seconds from trigger to delay reference
                point.  Maximum value depends on time/division setting.
```

Example

```
OUTPUT 707;":TIM:WIND:DEL 20E-9"
```

Query Syntax

```
:TIMebase:WINDow:DELay?
```

Returned Format

```
<wid_delay><NL>
```

Where:

```
<wid_delay> ::= current setting in seconds (exponential - NR3 format)
```

Example

```
DIM Dly$[50]
OUTPUT 707;":TIMEBASE:WINDOW:DELAY?"
ENTER 707;Dly$
PRINT Dly$
```

WINDow:RANGe (Timebase)

The :TIMEBASE:WINDOW:RANGE command sets the full-scale horizontal time in seconds for the second (expanded) timebase. The RANGE value is ten times the time per division of the second timebase.

The WINDOW:RANGE query returns the current range value.

Command Syntax

```
:TIMebase:WINDow:RANGe <range>
```

Where:

```
<range> ::= 100 ps to 50 s (depends on main sweep setting)
```

Example

```
OUTPUT 707;":TIM:WIND:RANG 100E-9"
```

Query Syntax

```
:TIMebase:WINDow:RANGe?
```

Returned Format

```
<range><NL>
```

Where:

```
<range> ::= 100 ps to 50 s (depends on main sweep setting)
           (exponential - NR3 format)
```

Example

```
DIM Wrng$[50]
OUTPUT 707;":TIMEBASE:WINDOW:RANGE?"
ENTER 707;Wrng$
PRINT Wrng$
```

17

Trigger Subsystem

The TRIGGER subsystem is used to define the conditions for a trigger. Many of the commands in the TRIGGER subsystem are used in more than one of the trigger modes. If the command is a valid command for a trigger mode that setting will be accepted. If the command is not valid for a trigger mode an error will be generated.

You must be sure that the instrument is in the proper trigger mode for the command being sent. One method of insuring this is to send the :TRIGGER:MODE command in the same program message as the parameter to be set. As an example, send:

```
":TRIGGER:MODETV;LEVEL 200 MV"
```

This will place the instrument in the TV Trigger Mode and set the trigger level to 200 mV. This is necessary because the LEVEL command is also valid for the other trigger modes.

The trigger modes are described on the next few pages prior to the command syntax.
Table 18-1 lists the different TRIGGER subsystem commands that are available for each trigger mode.

Note Auto or triggered mode is selected with the :TIMEBASE:MODE command.

Table 17-1. Valid Commands for Specific Trigger Modes

EDGE	PATTERN	STATE	DELAY	TV
CENTERED	CENTERED	CENTERED	CENTERED	CENTERED
HOLDOFF	CONDITION	CONDITION	CONDITION	CONDITION
LEVEL	HOLDOFF	HOLDOFF	DELAY	FIELD
SENSITIVITY	LEVEL	LEVEL	DELAY:SLOPE	HOLDOFF
SLOPE	LOGIC	LOGIC	DELAY:SOURCE	LEVEL
SOURCE	PATH	PATH	LEVEL	LINE
	SENSITIVITY	SENSITIVITY	LOGIC	OCCURRENCE
		SLOPE	OCCURRENCE	OCCURRENCE:SLOPE
		SOURCE	OCCURRENCE:SLOPE	POLARITY
			OCCURRENCE:SOURCE	QUALIFY
			PATH	SENSITIVITY
			QUALIFY	SOURCE
			SENSITIVITY	STANDARD
			SLOPE	
			SOURCE	

Edge Trigger Mode

The Edge Trigger Mode has the least number of parameters to be set. Use the following trigger commands to set up EDGE triggering.

In this mode you must set the trigger source, using the :TRIGGER:SOURCE command. This selects the channel that the oscilloscope will trigger on. The argument for this command is CHANNEL1 through CHANNEL4. Use the following trigger commands in the order presented to set up EDGE triggering.

The next thing that must be set in this mode is the trigger level. This value is set using the LEVEL command and can be set for each trigger source. The trigger level values that are set in this mode are used for all modes except TV Trigger Mode, or conversely, the Levels set in the PATTERN, STATE, or DELAY modes will set the Edge Levels as well. The trigger level is used in the PATTERN and STATE mode to define the voltage that determines if the input voltage is a logic high or a logic low for the logic triggers.

The next field to be set in the Edge Trigger Mode is the trigger sensitivity. This command is used to select noise reject on or off, and can be set for each trigger source.

The next field to be set in the Edge Trigger Mode is the actual edge that will create the trigger. This command is the SLOPE command and can be set to POSITIVE or NEGATIVE for each of the sources.

The last setting in this mode is the trigger holdoff value. This value is only used for the EDGE mode.

Pattern Trigger Mode

The Pattern Trigger Mode is used to define up to four patterns for the instrument to recognize, and then generate a trigger event. There are additional parameters in this mode that are not used in the Edge Trigger Mode and one parameter that is carried over from the edge mode. The one parameter that carriers over is LEVEL. If the LEVEL command is used in this mode it will also change the level value for that source in the Edge Trigger Mode. Use the following trigger commands in the order presented to setup PATTERN triggering.

The logic pattern for the Pattern Trigger Mode is set using the PATH and LOGIC commands. The PATH command specifies which of the four inputs is selected for the logic pattern. Once the path has been selected, the pattern can be set using the LOGIC command. The LOGIC command uses the arguments HIGH, LOW, and DONTCARE to set the "trigger on" bit pattern.

The next command is the CONDITION command, this command is used to select the "when" condition that must be satisfied before a trigger event is generated. This command is used in several of the trigger modes, therefore it has parameters that are not valid in this mode. The valid parameters for the CONDITION command in the Pattern Trigger Mode are: ENTER, EXIT, GT, LT, and RANGE.

When the command :TRIGGER:CONDITION ENTER or :TRIGGER:CONDITION EXIT is sent, the Entered or Exited parameter is set. When the GT or LT option is used, a time value must be sent to define the limit. When the RANGE option is used, two time values must be sent to define the lower and upper limit. The correct syntax for the RANGE option is :TRIGGER:CONDITION RANGE,<range_low>,<range_high>.

Also, in the Pattern Trigger Mode, you can set the holdoff time using the :TRIGGER:HOLDOFF command.

State Trigger Mode

When the State Trigger Mode is selected the :TRIGGER:SOURCE command is used to select the clock source. The syntax for selecting the clock source is :TRIGGER:SOURCE<channel>.

After the clock source is selected, the correct edge for the clock can be selected using the :TRIGGER:SLOPE command which can be set to NEGATIVE or POSITIVE.

Next, the :TRIGGER:PATH command can be used with the :TRIGGER:LOGIC command to set the three bit logic pattern to qualify the clock trigger. These commands could be sent using the following syntax ":TRIGGER:PATH CHAN2;:TRIGGER:LOGIC LOW", or ":TRIGGER:PATH CHAN2;LOGIC LOW".

The :TRIGGER:CONDITION command in the State Trigger Mode will set the "is/is not present" state using the parameters TRUE for "is present" and FALSE for "is not present."

In this mode, a holdoff value can be set as in most other modes.

Delay Trigger Mode

In the Delay Trigger Mode, the :TRIGGER:QUALIFY command can be used to select the EDGE, PATTERN, or STATE mode as a qualifier. When the EDGE qualifier is selected, all Edge parameters and commands can be used to set the source and slope. When the PATTERN qualifier is selected, all Pattern commands can be used to set the pattern mode parameters. When the STATE qualifier is selected, all State commands can be used to set the state mode parameters.

The next settings are the delay settings. The DELAY command is used to set the Time or Count parameter and the amount of delay. To set the delay to time, use the command :TRIGGER:DELAY TIME,<time>, and to set the delay to count, use the command :TRIGGER:DELAY EVENT<number_events>.

If the trigger delay is set to Event (count), you can then set the delay source and slope. To set the delay source, use the command ":TRIGGER:DELAY:SOURCE CHANNEL2", and to set the delay slope, use the command ":TRIGGER:DELAY:SLOPE POSITIVE".

Next is the "trigger on" field. The values within this field are set with the OCCURRENCE command. To set the number of occurrences, use the command syntax ":TRIGGER:OCCURRENCE 3333". To set the source for the number of occurrences, use the command syntax ":TRIGGER:OCCURRENCE:SOURCE CHANNEL2", and to set the slope of the trigger occurrence, use the command syntax ":TRIGGER:OCCURRENCE:SLOPE NEGATIVE".

TV Trigger Mode

The TV Trigger Mode is used for triggering on clamped television signals. This mode will allow you to select one of the TV signal frames and one of the lines within that frame.

Once the TV Trigger Mode has been selected, the Television Signal Standard can be selected using the :TRIGGER:STANDARD command. The three parameters for this command are 525, 625, and USER. Any of these modes allow you to select the source of the trigger signal and the trigger level.

The source is set by sending the SOURCE command. The SOURCE command allows the selection of channel 1 through 4 using the command ":TRIGGER:SOURCE CHANNEL2".

The trigger level is set by sending the command ":TRIGGER:LEVEL <value>".

With the standard set to 525 or 625, the commands that can be used are POLARITY, FIELD, and LINE. The POLARITY command can accept NEGATIVE and POSITIVE as its parameters and sets the edge for the trigger. The FIELD command uses 1 and 2 for its parameters which select the first or second field of the television signal. The LINE command parameters are different for the two standards, refer to the command to determine the correct values.

The HOLDOFF value can also be set in the TV Trigger Mode, as in all modes.

When the "user defined" standard is selected, the source and level are set in the same manner as before.

The QUALIFY command is used to set the "qualify on" field. This command uses the parameters HIGH and LOW.

The :TRIGGER:CONDITION RANGE command sets the greater than and less than time values. In order to actually generate a trigger, the qualified conditions must be met within the specified time. To set the time values, send ":TRIGGER:CONDITION RANGE,<gt_value>, <lt_value>".

The next field "trigger on" is set with OCCURRENCE and :OCCURRENCE:SLOPE commands. To set the number of occurrences, send the command ":TRIGGER:OCCURRENCE <number>", and to set the slope for the occurrences, send the command ":TRIGGER:OCCURRENCE:SLOPE POSITIVE". The slope command can also use NEGATIVE as a parameter.

The description for each of the commands will tell you in which modes that command is valid.

Refer to figure 17-1 for TRIGGER subsystem commands syntax diagram.

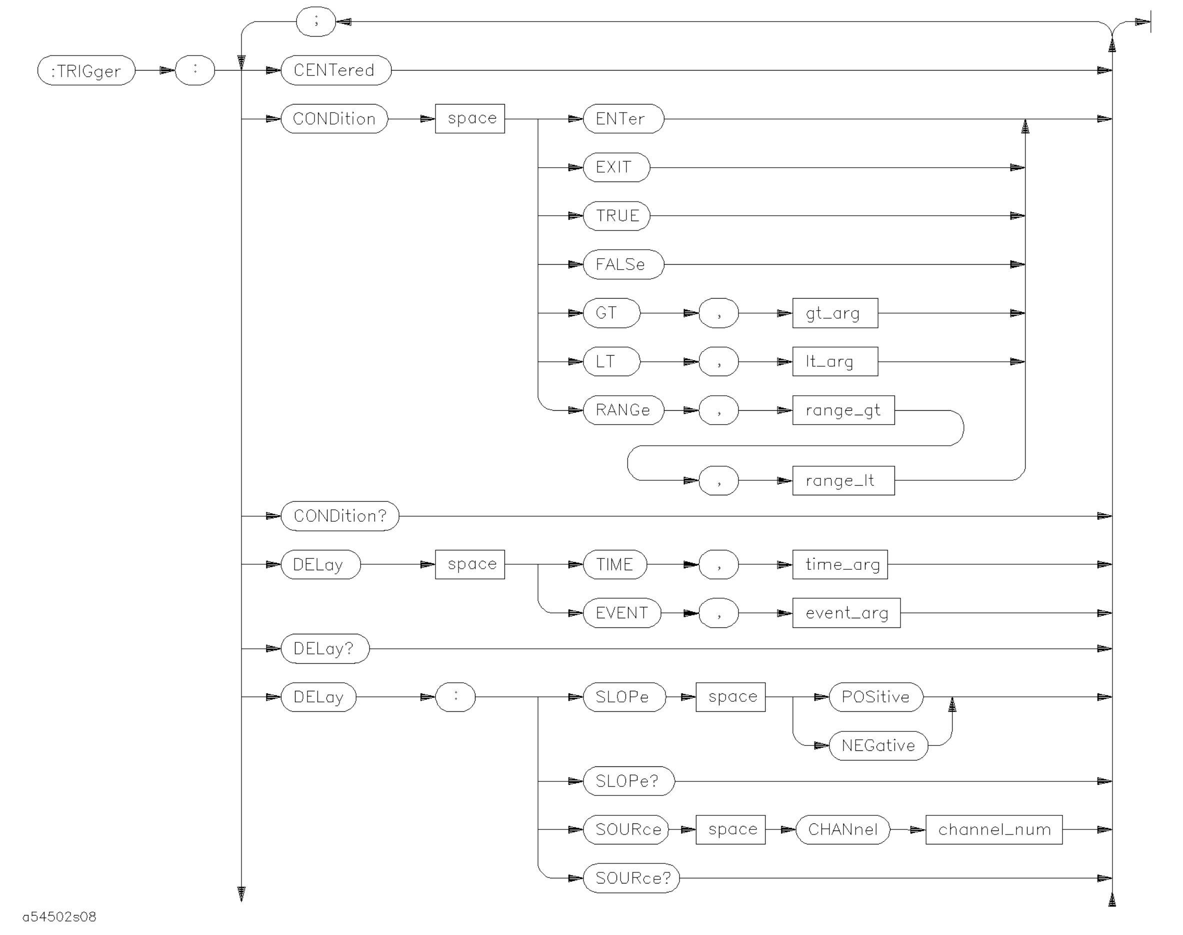


Figure 17-1. TRIGGER Subsystem Commands Syntax Diagram

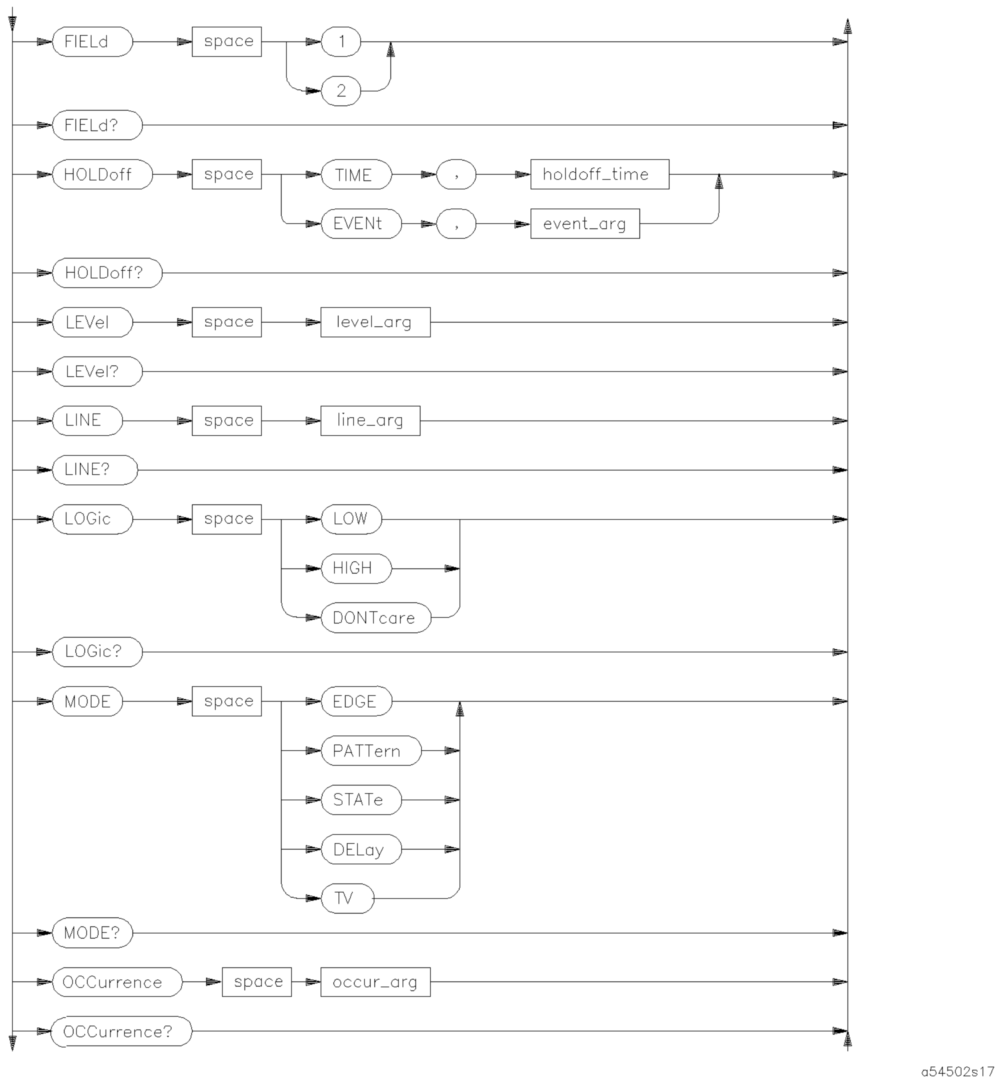


Figure 17-1. TRIGGER Subsystem Commands Syntax Diagram (continued)

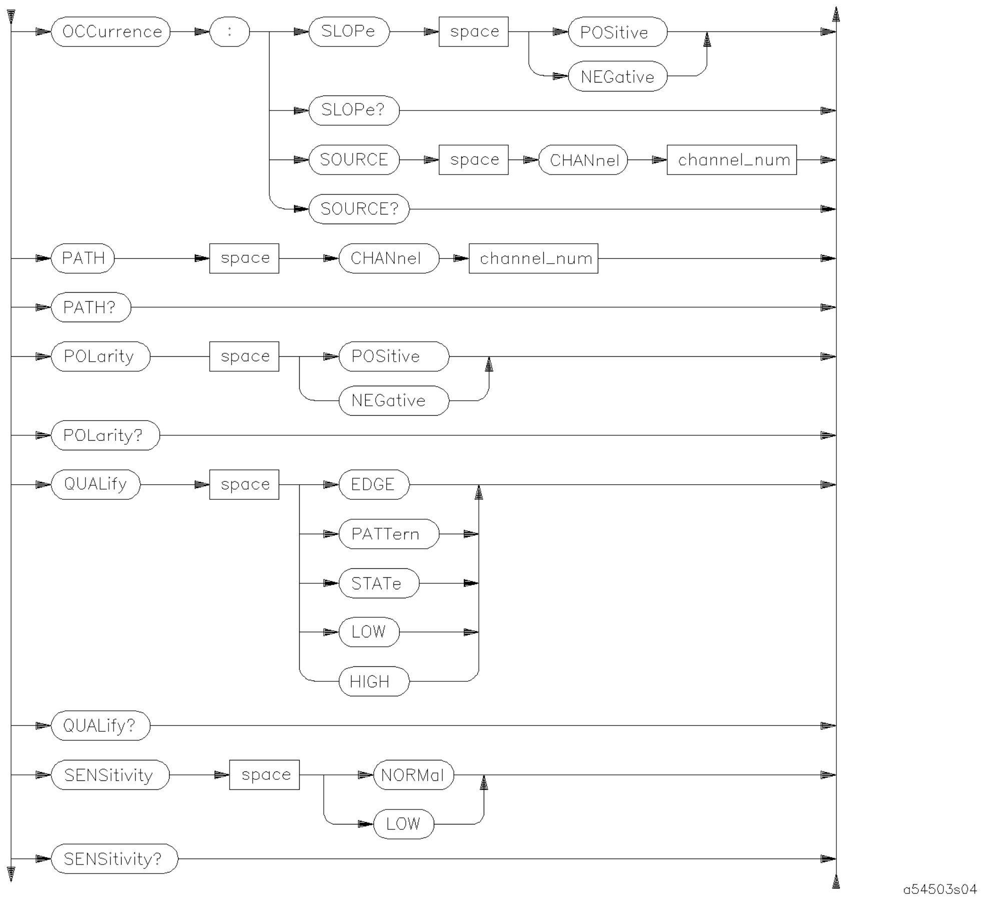
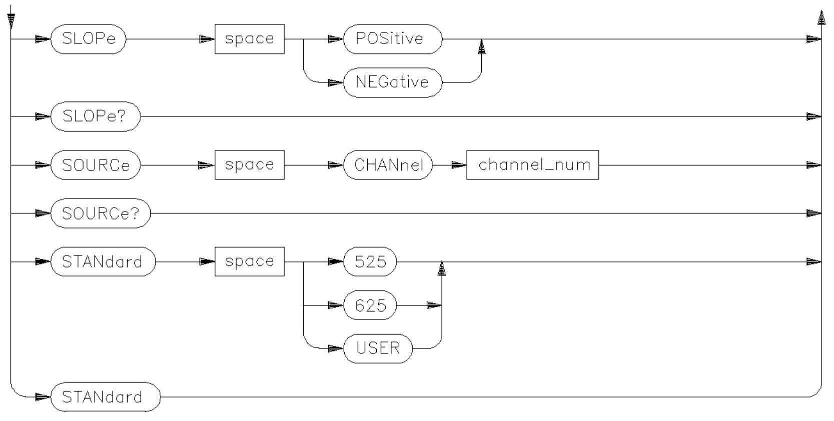


Figure 17-1. TRIGGER Subsystem Commands Syntax Diagram (continued)



channel_num	=	integer, 1 through 4
event_arg	=	integer, 1 to 16000000
gt_arg	=	time value, 20 ns to 160 ms
holdoff_time	=	time value, 40 ns to 320 ms
level_arg	=	real number (specifies trigger level in volts)
line_arg	=	integer, 1 to 625 (depends on video STANDARD selected)
lt_arg	=	time value, 20 ns to 160 ms
range_gt	=	time value, 20 ns to 159.999 ms (value must be less than range_lt)
range_lt	=	time value, 30 ns to 160 ms (value must be greater than range_gt)
time_arg	=	time value, 30 ns to 160 ms
occur_arg	=	integer, 1 to 16000000

a54503s10

Figure 17-1. TRIGGER Subsystem Commands Syntax Diagram (continued)

CENTered

The :TRIGGER:CENTERED command sets the trigger level to the current vertical offset value for the channel selected.

Note To return to the ADJUST mode, specify a level with the :TRIGGER:LEVEL command.

Command Syntax

```
:TRIGger:CENTered
```

Example

```
OUTPUT 707;":TRIGGER:CENTERED"
```

CONDition

The :TRIGGER:CONDITION command is valid in the PATTERN, STATE, DELAY, and TV Trigger Modes. The function of the CONDITION command in each of these modes is described below.

Time values entered using this command are rounded to the nearest 10 ns.

In the Pattern Trigger Mode, the valid arguments for the CONDITION command are ENTER, EXIT, GT, LT, RANGE.

In the Pattern Trigger Mode, the CONDITION command is used to specify if the trigger will be generated upon entry to the specified logic pattern, upon exiting the specified logic pattern, or if the pattern must be present for a specified amount of time. The time in the pattern trigger mode can be specified to be greater than a value (GT), less then a value (LT), or between two values (RANGE).

In the State Trigger Mode, the valid parameters for the CONDITION command are TRUE (is present) and FALSE (is not present).

In the Delay Trigger Mode, the CONDITION command is valid when PATTERN or STATE is selected as the qualifier. All arguments for this command that are valid in the Pattern or State Trigger Modes are valid here.

In the TV Trigger Mode, the CONDITION command is used to set the range of time values for the trigger to occur. This command is only valid in the “user defined” mode.

The CONDITION query returns the currently selected condition, for the currently selected mode.

Command Syntax

```
:TRIGger:CONDition <argument>
```

Where, in PATTERN or DELAY:PATTERN mode:

```
<argument> ::= {ENTer|EXIT|GT,<value>|LT,<value>|
                        RANGe,<range_gt>,<range_lt>}
```

Where, in STATE or DELAY:STATE mode:

```
<argument> ::= {TRUE|FALSe}
```

Where, in TV mode:

```
<argument> ::= RANGe,<range_gt>,<range_lt>
```

Where:

```
<value> ::= 20 ns to 160 ms
<range_gt> ::= 20 ns to 159.999 ms (must be less than range_lt)
<range_lt> ::= 30 ns to 160 ms (must be greater than range_gt)
```

Example

```
OUTPUT 707;":TRIG:COND RANGE,22ms,33ms"
```

Query Syntax

```
:TRIGger:CONDition?
```

Returned Format

```
<argument><NL>
```

Where, in PATTERN or DELAY PATTERN mode:

```
<argument> ::= {ENTer|EXIT|GT,<value>|
LT,<value>|RANGe,<range_gt>,<range_lt>}
```

Where, in STATE or DELAY STATE mode:

```
<argument> ::= {TRUE|FALSe}
```

Where, in TV mode:

```
<argument> ::= RANGe,<range_gt>,<range_lt>
```

Where:

```
<value> ::= 20 ns to 160 ms
<range_gt> ::= 20 ns to 159.999 ms (must be less than range_lt)
<range_lt> ::= 30 ns to 160 ms (must be greater than range_gt)
```

Example

```
DIM Con$[50]
OUTPUT 707;"CONDITION?"
ENTER 707;Con$
PRINT Con$
```

DELay

The :TRIGGER:DELAY command is valid only in the Delay Trigger Mode. This command allows you to set a delay value in either time or number of events. In the time delay mode, this command specifies the delay value in seconds. In the events delay mode, this command specifies the delay value in number of trigger events.

The DELAY query returns the current delay setting.

Command Syntax

```
:TRIGger:DELay {TIME,<time_value>|EVENt,<event_value>}
```

Where:

```
<time_value> ::= time of delay from 30 ns to 160 ms
<event_value> ::= number of events from 1 to 16000000
```

Example

```
OUTPUT 707;":TRIGGER:DELAY TIME,1.23E-01"
```

Query Syntax

```
:TRIGger:DELay?
```

Returned Format

```
{TIME,<time_value>|EVENt,<event_value>}<NL>
```

Where:

```
<time_value> ::= time of delay from 30 ns to 160 ms
<event_value> ::= number of events from 1 to 16000000
```

Example

```
DIM Value$[50]
OUTPUT 707;":TRIG:DELAY?"
ENTER 707;Value$
PRINT Value$
```

DELay:SLOPe

The :TRIGGER:DELAY:SLOPE command sets the edge that will be counted by the delay command. The parameters for this command are NEGATIVE or POSITIVE. This command is valid in the Delay Trigger Mode.

The DELAY:SLOPE query returns the current delay slope.

Command Syntax

```
:TRIGger:DELay:SLOPe {POSitive|NEGative}
```

Example

```
OUTPUT 707;":TRIG:DEL:SLOP POS"
```

Query Syntax

```
:TRIGger:DELay:SLOPe?
```

Returned Format

```
{POSitive|NEGative}<NL>
```

Example

```
DIM Tos$[50]
OUTPUT 707;"TRIGGER:DELAY:SLOP?"
ENTER 707;Tos$
PRINT Tos$
```

DELay:SOURce

The :TRIGGER:DELAY:SOURCE command sets the edge that will be counted by the delay command. The parameters for this command are CHANNEL1 through CHANNEL4. This command is only valid in the Delay Trigger Mode. The DELAY:SOURCE query returns the source of the delay in the Delay Trigger Mode.

Command Syntax

```
:TRIGger:DELay:SOURce {CHANnel1|CHANnel2|CHANnel3|CHANnel4}
```

Example

```
OUTPUT 707;":TRIG:DEL:SOURCE CHANNEL2"
```

Query Syntax

```
:TRIGger:DELay:SOURce?
```

Returned Format

```
{CHANnel1|CHANnel2|CHANnel3|CHANnel4}<NL>
```

Example

```
DIM Tos$[50]
OUTPUT 707;"TRIGGER:DELAY:SOUR?"
ENTER 707;Tos$
PRINT Tos$
```

FIELd

The :TRIGGER:FIELD command is only valid in the TV Trigger Mode. This command is used to select the field of the TV signal when the STANDARD is set to 525 or 625. The only valid parameters for this command are 1 or 2.

The FIELD query returns the current setting of the FIELD command.

Command Syntax

```
:TRIGger:FIELd {1|2}
```

Example

```
OUTPUT 707;"TRIGGER:FIEL 2"
```

Query Syntax

```
:TRIGger:FIELd?
```

Returned Format

```
{1|2}<NL>
```

Example

```
DIM F$[50]
OUTPUT 707;":TRIG:FIELD?"
ENTER 707;"F$
PRINT F$
```

HOLDoff

The :TRIGGER:HOLDOFF command is valid in the Edge, Pattern, State, and TV Trigger Modes. This command will allow a holdoff by time value to be entered.

The HOLDOFF query returns the value of the holdoff for the current mode.

Command Syntax

```
:TRIGger:HOLDoff {TIME,<holdoff_value>|EVENT,<event_arg>}
```

Where:

```
<holdoff_value> ::= 40 ns to 320 ms rounded to nearest 20ns
                    increment. <event_arg> ::= 1 to 16000000
```

Examples

```
OUTPUT 707;":TRIGGER:HOLDOFF TIME,216 US"
OUTPUT 707;":TRIGGER:HOLDOFF EVENT,10"
```

Query Syntax

```
:TRIGger:HOLDoff?
```

Returned Format

```
{TIME,<holdoff_value>|EVENt,<event_arg>}<NL>
```

Where:

```
<holdoff_value> ::= 40 ns to 320 ms (exponential - NR3 format)
<event_arg> ::= 1 to 16000000 (integer - NR1 format)
```

Example

```
DIM Ho$[50]
OUTPUT 707;":TRIGGER:HOLD?"
ENTER 707;Ho$
PRINT Ho$
```

LEVel

The :TRIGGER:LEVEL command sets the trigger level voltage of the active trigger. This command can be sent in any mode, however only two separate levels are stored. One value is kept for the TV Trigger Mode and another value is kept for all other modes. If you are in the Pattern Trigger Mode and set a trigger level value, that level will also be used for the Edge, State, and Delay Trigger Modes.

The LEVEL query returns the trigger level of the current trigger mode.

Command Syntax

```
:TRIGger:LEVel <level>
```

Where:

```
<level> ::= -140.0 volts to +140.0 volts (maximum value
            depends on volts/division and offset settings).
```

Examples

```
OUTPUT 707;":TRIGGER:LEVEL .30"
OUTPUT 707;":TRIGGER:LEV 300MV"
OUTPUT 707;":TRIG:LEV 3E-1"
```

Refer to Appendix B for the syntax of using values with multipliers.

Query Syntax

```
:TRIGger:LEVel?
```

Returned Format

```
<level><NL>
```

Where:

```
<level> ::= trigger level in volts (exponential - NR3 format)
```

Example

```
DIM Tlevel$[30]
OUTPUT 707;":TRIGGER:LEVEL?"
ENTER 707;Tlevel$
PRINT Tlevel$
```

LINE

The :TRIGGER:LINE command is valid in the TV Trigger Mode when the STANDARD selected is 525 or 625. If one of the these standards is selected when the TV Trigger Mode is entered the line value will be set in that standard and selected field.

The LINE query returns the current line of the selected standard.

Command Syntax

```
:TRIGger:LINE <line_number>
```

Where:

```
<line_number> ::= 1 to 625 (depends on STANDARD and FIELD selection).
```

Example

```
OUTPUT 707;":TRIG:LINE 22"
```

Query Syntax

```
:TRIGger:LINE?
```

Returned Format

```
<line_number><NL>
```

Where:

```
<line_number> ::= 1 to 625 (depends on STANDARD and FIELD selection).
                  (integer - NR1 format)
```

Example

```
DIM Ln$[50]
OUTPUT 707;":TRIGGER:LINE?"
ENTER 707;Ln$
PRINT Ln$
```

LOGic

The :TRIGGER:LOGIC command is valid in the Pattern and State Trigger Modes, as well as the Delay Trigger Mode (when qualifying by PATTERN or STATE). The LOGIC command is used to specify the relation between the signal and the defined voltage level that must exist before that part of the pattern is considered valid. If the signal on a selected path is greater than the trigger level, that signal is considered HIGH. If it is less than the trigger level, it is considered LOW.

The LOGIC query returns the last specified logic level of the currently enabled path.

Command Syntax

```
:TRIGger:LOGic {HIGH|LOW|DONTcare}
```

Example

```
OUTPUT 707;":TRIG:LOGIC DONT"
```

Query Syntax

```
:TRIGger:LOGic?
```

Returned Format

```
{HIGH|LOW|DONTcare}<NL>
```

Example

```
DIM L$[50]
OUTPUT 707;":TRIGGER:LOGIC?"
ENTER 707;L$
PRINT L$
```

MODE

The :TRIGGER:MODE command selects the trigger mode. The MODE command can be sent from any trigger mode.

The MODE query returns the currently selected trigger mode.

Command Syntax

```
:TRIGger:MODE {EDGE|PATTern|STATe|DELay|TV}
```

Example

```
OUTPUT 707;":TRIGGER:MODE PATT"
```

Query Syntax

```
:TRIGger:MODE?
```

Returned Format

```
{EDGE|PATTern|STATe|DELay|TV}<NL>
```

Example

```
DIM Mode$[50]
OUTPUT 707;":TRIGGER:MODE?"
ENTER 707;Mode$
PRINT Mode$
```

OCCurrence

The :TRIGGER:OCCURRENCE command sets the number of trigger events that must occur before the oscilloscope sweep is actually triggered. This command is valid in the Delay Trigger Mode and the TV Trigger Mode.

The OCCURRENCE query returns the current value of occurrence if the oscilloscope is in the Delay Trigger Mode, or in the TV Trigger Mode with USER DEFINED selected.

Command Syntax

```
:TRIGger:OCCurrence <occ_number>
```

Where:

```
<occ_number> ::= 1 to 16000000
```

Example

```
OUTPUT 707;":TRIGGER:OCC 14"
```

Query Syntax

```
:TRIGger:OCCurrence?
```

Returned Format

```
<occ_number><NL>
```

Where:

```
<occ_number> ::= 1 to 16000000 (integer - NR1 format)
```

Example

```
DIM Oc$[50]
OUTPUT 707;":TRIGGER:OCCURRENCE?"
ENTER 707;Oc$
PRINT Oc$
```

OCCurrence:SLOPe

The :TRIGGER:OCCURRENCE:SLOPE command sets the edge that will be counted by the OCCURRENCE command. The parameters for this command are NEGATIVE or POSITIVE. This command is valid in the Delay Trigger Mode and the TV Trigger Mode.

The OCCURRENCE:SLOPE query returns the slope of the current mode.

Command Syntax

```
:TRIGger:OCCurrence:SLOPe {POSitive|NEGative}
```

Example

```
OUTPUT 707;":TRIG:OCC:SLOP POS"
```

Query Syntax

```
:TRIGger:OCCurrence:SLOPe?
```

Returned Format

```
{POSitive|NEGative}<NL>
```

Example

```
DIM Tos$[50]
OUTPUT 707;"TRIGGER:OCCURRENCE:SLOP?"
ENTER 707;Tos$
PRINT Tos$
```

OCCurrence:SOURce

The :TRIGGER:OCCURRENCE:SOURCE command sets the edge that will be counted by the occurrence command. The parameters for this command are CHANNEL1 through CHANNEL4. This command is valid in the Delay Trigger Mode.

The OCCURRENCE:SOURCE query returns the source of the occurrence in the Delay Trigger Mode.

Command Syntax

```
:TRIGger:OCCurrence:SOURce {CHANnel1|CHANnel2|CHANnel3|CHANnel4}
```

Example

```
OUTPUT 707;":TRIG:OCC:SOURCE CHANNEL2"
```

Query Syntax

```
:TRIGger:OCCurrence:SOURce?
```

Returned Format

```
{CHANnel1|CHANnel2|CHANnel3|CHANnel4}<NL>
```

Example

```
DIM Tos$[50]
OUTPUT 707;"TRIGGER:OCCURRENCE:SOUR?"
ENTER 707;Tos$
PRINT Tos$
```

PATH

The :TRIGGER:PATH command is valid in the Pattern Trigger Mode, State Trigger Mode, and Delay Trigger Mode when "qualify on" pattern or state is selected. This command selects a pattern bit as the source for future logic commands.

The PATH query returns the current trigger source of the present mode.

Command Syntax

```
:TRIGger:PATH <path_name>
```

Where:

```
<path_name> ::= {CHANnel1|CHANnel2|CHANnel3|CHANnel4}
```

Example

```
OUTPUT 707;":TRIGGER:PATH CHANNEL2"
```

Query Syntax

```
:TRIGger:PATH?
```

Returned Format

```
CHANnel{1|2|3|4}<NL>
```

Example

```
DIM Tp$[50]
OUTPUT 707;":TRIG:PATH?"
ENTER 707;Tp$
PRINT Tp$
```

POLarity

The :TRIGGER:POLARITY command is valid in the TV Trigger Mode. It sets the polarity for the trigger when the STANDARD is set to 525 or 625. The valid parameters for this command are POSITIVE and NEGATIVE.

The POLARITY query will return the current polarity setting.

Command Syntax

```
:TRIGger:POLarity {POSitive|NEGative}
```

Example

```
OUTPUT 707;":TRIGGER:POL NEGATIVE"
```

Query Syntax

```
:TRIGger:POLarity?
```

Returned Format

```
{POSitive|NEGative}<NL>
```

Example

```
DIM Tp$[50]
OUTPUT 707;":TRIG:POL?"
ENTER 707;Tp$
PRINT Tp$
```

QUALify

The :TRIGGER:QUALIFY command is valid in the Delay and TV Trigger Mode. The parameters for this command when in the Delay Trigger Mode are:

- EDGE
- PATTERN
- STATE

The parameters for this command when in the TV Trigger Mode are:

- LOW
- HIGH

The QUALIFY query returns the current setting of the QUALIFY command in the currently selected mode.

Command Syntax

```
:TRIGger:QUALify <qualify_parameter>
```

Where, in Delay Trigger Mode:

```
<qualify_parameter> ::= {EDGE|PATTern|STATe}
```

Where, in TV Trigger Mode:

```
<qualify_parameter> ::= {LOW|HIGH}
```

Example

```
OUTPUT 707;":TRIGGER:QUALIFY PATT"
```

Query Syntax

```
:TRIGger:QUALify?
```

Returned Format

```
{EDGE|PATTern|STATe|LOW|HIGH}<NL>
```

Example

```
DIM Tq$[50]
OUTPUT 707;":TRIG:QUALIFY?"
ENTER 707;Tq$
PRINT Tq$
```

SENSitivity

The :TRIGGER:SENSITIVITY command sets the trigger sensitivity for the selected source. NORMAL corresponds to noise reject off and LOW corresponds to noise reject on.

The SENSITIVITY query returns the current sensitivity for the currently selected source.

```
:TRIGger:SENSitivity {NORMal|LOW}
```

Example

```
OUTPUT 707;":TRIGGER:SENSITIVITY LOW"
```

Query Syntax

```
:TRIGger:SENSitivity?
```

Returned Format

```
{NORMal|LOW}<NL>
```

Example

```
OUTPUT 707;":TRIG:SENS?"
ENTER 707;Sens$
PRINT Sens$
```

SLOPe

The :TRIGGER:SLOPE command specifies the slope of the edge for the trigger. The SLOPE command is valid in the Edge Trigger Mode, State Trigger Mode, and Delay Trigger Mode when EDGE or STATE is selected as the qualifier.

The SLOPE query returns the current slope for the currently selected trigger mode.

Command Syntax

```
:TRIGger:SLOPe {NEGative|POSitive}
```

Example

```
OUTPUT 707;":TRIGGER:SLOPE POSITIVE"
```

Query Syntax

```
:TRIGger:SLOPe?
```

Returned Format

```
{POSitive|NEGative}<NL>
```

Example

```
DIM Ts$[50]
OUTPUT 707;":TRIG:SLOP?"
ENTER 707;Ts$
PRINT Ts$
```

SOURce

The :TRIGGER:SOURCE command selects the channel that actually produces the trigger. The SOURCE command is valid in the Edge Trigger Mode, State Trigger Mode, Delay Trigger Mode, and TV Trigger Mode. In the Delay Trigger Mode this command is valid when EDGE or STATE is selected as the qualifier.

The SOURCE query returns current source for the selected trigger mode.

Command Syntax

```
:TRIGger:SOURce {CHANnel1|CHANnel2|CHANnel3|CHANnel4}
```

Example

```
OUTPUT 707;":TRIGGER:SOURCE CHAN2"
```

Query Syntax

```
:TRIGger:SOURce?
```

Returned Format

```
{CHANnel1|CHANnel2|CHANnel3|CHANnel4}<NL>
```

Example

```
DIM Src$[30]
OUTPUT 707;":TRIGGER:SOURCE?"
ENTER 707;Src$
PRINT Src$
```

STANdard

The :TRIGGER:STANDARD command selects the television signal standard to be used in the TV Trigger Mode. The valid parameters for this command 525, 625, and USER (defined).

The STANDARD query returns the currently selected standard.

Command Syntax

```
:TRIGger:STANdard {525|625|USER}
```

Example

```
OUTPUT 707;":TRIGGER:STAN USER"
```

Query Syntax

```
:TRIGger:STANdard?
```

Returned Format

```
{525|625|USER}<NL>
```

Example

```
DIM Ts$[50]
OUTPUT 707;":TRIG:STANDARD?"
ENTER 707;Ts$
PRINT Ts$
```

18

Waveform Subsystem

The WAVEFORM subsystem is used to transfer waveform data between a controller and the HP 70703A's waveform memories. The waveform record is actually contained in two portions, the waveform data and the preamble. The waveform data is the actual data acquired for each point in the specified source. The preamble contains the information for interpreting the waveform data. This includes the number of points acquired, format of acquired data, and type of acquired data. The preamble also contains the X and Y increments, origins, and references for the acquired data, so that the raw data can be translated to time and voltage values.

The values set in the preamble are determined when the :DIGITIZE command is executed. The preamble values are based on the settings of variables in the ACQUIRE subsystem. Although the preamble values can be changed with a controller, the way the data was acquired cannot be changed. Changing the preamble values cannot change the type of data that was actually acquired, the number of points actually acquired, and so on. Therefore, extreme caution must be used when changing any waveform preamble values to ensure the data will still be useful. For example, setting POINTS in the preamble to a value different from the actual number of points in the waveform will result in inaccurate data.

The waveform data and preamble must be read (by the controller) or sent (to the HP 70703A) with two separate commands, DATA and PREAMBLE.

Sending consecutive :DIGITIZE commands may improve the data throughput. Refer to the Root Level Command :DIGITIZE for more information.

Data Acquisition Types

There are three types of waveform acquisition that can be selected with the :ACQUIRE:TYPE command. The three types are NORMAL, AVERAGE, and ENVELOPE. When the data is acquired using the :DIGITIZE command the data is placed in the channel buffer of the specified source.

After a :DIGITIZE command, the instrument is stopped. If the instrument is restarted, the data acquired with the :DIGITIZE command will be overwritten

Note The on-screen time is divided into a specific number of horizontal time points as defined by the :ACQUIRE:POINTS command. Each of these increments in time is referred to as a time bucket with each time bucket having a fixed time associated with it.

Normal

Normal data consists of the last data point (hit) in each time bucket. This data is transmitted over HP-IB in a linear fashion starting with time bucket 0 and going through time bucket n−1, where n is the number returned by the :WAVEFORM:POINTS query. Time buckets that don't have data in them return −1. Only the magnitude values of each data point are transmitted, the time values correspond to the position in the data array. The first voltage value corresponds to the first time bucket on the left of the active waveform and the last value corresponds to the next to last time bucket on the right side of the active waveform.

Average

Average data consists of the average of the first n hits in a time bucket, where n is the value returned by the :ACQUIRE:COUNT query. Time buckets that have fewer than n hits return the average of what data they do have. If the :ACQUIRE:COMPLETE parameter is set to 100%, then each time bucket must contain the number of data hits specified with the :ACQUIRE:COUNT command. Again, if a time bucket doesn't have any data in it, it will return −1. This data is transmitted over the HP-IB in a linear fashion starting with time bucket 0 and proceeding through time bucket n−1, where n is the number returned by the :WAVEFORM:POINTS query. The first value corresponds to a point at the left side of the active waveform and the last value is one point away from the right side of the active waveform.

Envelope

Envelope data consists of two arrays of data, one containing the minimum of the first n hits in each time bucket and the other containing the maximum of the first n hits in each time bucket, where n is the value returned by the :ACQUIRE:COUNT query. If a time bucket does not have any hits in it, then −1 is returned for both the minimum and maximum values. The two arrays are transmitted one at a time over the HP-IB linearly, starting with time bucket 0 (on the left side of the active waveform) and proceeding through time bucket m−1, where m is the value returned by the :WAVEFORM:POINTS query. The array with the minimum values is sent first. The first value of each array corresponds to the data point on the left of the active waveform. The last value is one data point away from the right side of the active waveform.

The data is transferred from the channel buffer to a controller using the :WAVEFORM:DATA query.

Data is transferred into the instrument from a controller using the :WAVEFORM:DATA command. Envelope data can be transferred into Waveform Memories 1 and 3, if WMEMORY 1 is specified as the source, or Waveform Memories 2 and 4, if WMEMORY 2 is specified as the source. The data is then transferred as two arrays. If Waveform Memory 1 is specified as the source, the first array is transferred into Waveform Memory 1 and the second array is transferred into Waveform Memory 3. If Waveform Memory 2 is specified as the source, the first array is transferred into Waveform Memory 2 and the second array is transferred into Waveform Memory 4. The data type is then changed to normal for each of the waveform memories.

Data Conversion

Data sent from the HP 70703A is raw data and must be scaled for useful interpretation. The values used to interpret the data are the X and Y references, X and Y origins, and X and Y increments. These values are read from the waveform preamble.

Conversion from Data Value to Voltage

The formula to convert a data value from waveform memories 1 through 4 to a voltage value is:

voltage = [(data value - yreference)*yincrement] + yorigin

Conversion from Data Value to Time

The time value of a data point can be determined by the position of the data point. As an example, the third data point sent with XORIGIN = 16 ns, XREFERENCE = 0, and XINCREMENT = 2 ns. Using the formula:

time = [(data point number - xreference)*xincrement] + xorigin

would result in the following calculation:

time = [(3 - 0) * 2 ns] + 16 ns = 22 ns.

Data Format for HP-IB Transfer

There are three formats for transferring waveform data over the HP-IB. These formats are WORD, BYTE, and COMPRESSED.

WORD, BYTE, and COMPRESSED formatted waveform records are transmitted using the arbitrary block program data format specified in IEEE 488.2.

When you use this format, the ASCII character string “#8<DD...D” is sent before the actual data. The 8 indicates how many <D>’s will follow. The <D>’s are ASCII numbers, which indicate how many data bytes will follow.

For example, if 512 points were acquired the Block Header “#800000512” would be sent. The 8 indicates that eight length bytes follow, 512 indicates that 512 data bytes (binary) follow.

WORD Format

In the WORD format, the number of data bytes is twice the number of words (data points). The number of data points is the value returned by the :WAVEFORM:POINTS? query. The number of data bytes is followed by a sequence of bytes representing the data points, with the most significant byte of each word transmitted first. In this format the data is shifted so that the most significant bit after the sign bit contains the most significant bit of the data. If there is a hole in the data, it will be represented by the 16-bit value of −1. The range of data in the WORD format is from 0 to 32640.

WORD format is useful in applications where the information is read directly into an integer array in a controller.

WORD formatted data returns the most accurate data values and greatest resolution.

BYTE Format

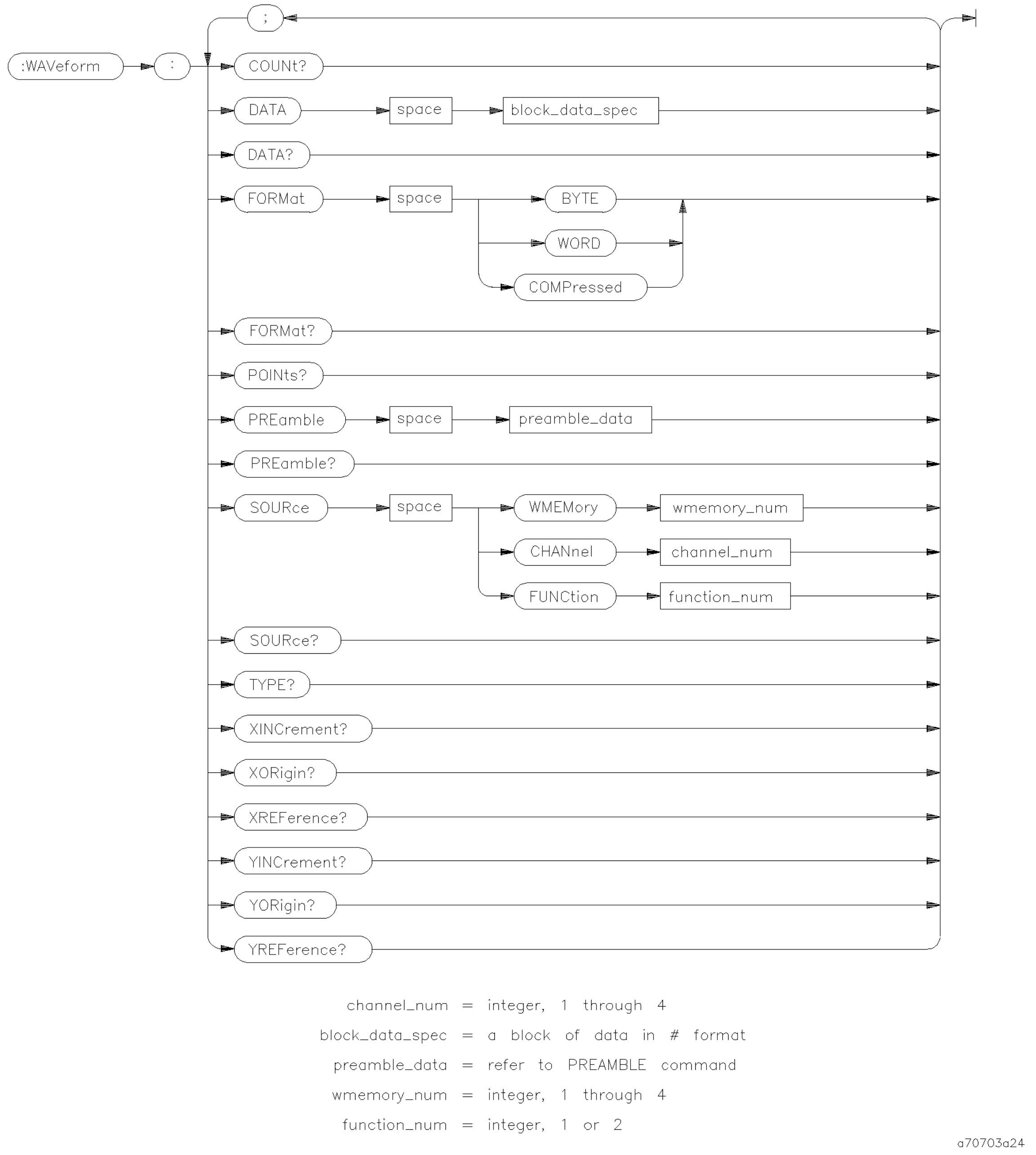
BYTE format allows only seven bits to be used to represent the voltage values, with the first bit being the sign bit. If there is a hole in the data, it is represented by a value of −1.

BYTE formatted data will transfer over the HP-IB faster than WORD formatted data, since one byte per point is transferred in BYTE format and two bytes per point are transferred in WORD format. BYTE formatted data has less resolution than WORD formatted data.

COMPRESSED Format

The number of bytes transmitted when the format is COMPRESSED is the same as the value returned by the :WAVEFORM:POINTS? query.

Eight bits of resolution are retained in the COMPRESSED format. So that a hole in the data may be represented, a data value of 255 is mapped to 254, and 255 is used to represent a hole. This mode will give greater vertical precision than BYTE formatted data, with faster transfer times than WORD formatted data, but will probably require more time once transferred to be unpacked.



channel_num	=	integer, 1 through 4
block_data_spec	=	a block of data in # format
preamble_data	=	refer to PREAMBLE command
wmemory_num	=	integer, 1 through 4
function_num	=	integer, 1 or 2

Figure 18-1. Waveform Subsystem Commands Syntax Diagram

COUNt

The :WAVEFORM:COUNT query will always return a 1 in this instrument. This query is only included, in this instrument, for compatibility with other Hewlett-Packard instruments.

The value returned for this query has no meaning for the HP 70703A.

Query Syntax

```
:WAVeform:COUNt?
```

Returned Format

```
1<NL>
```

Example

```
DIM Cnt$[50]
OUTPUT 707;":WAVEFORM:COUNT?"
ENTER 707;Cnt$
PRINT Cnt$
```

DATA

The :WAVEFORM:DATA command causes the instrument to accept a waveform data record over the HP-IB and store it in the previously specified waveform memory. The waveform memory is specified with a :WAVEFORM:SOURCE command. Only waveform memories may have waveform data sent to them.

Note The format of the data being sent must match the format previously specified by the waveforms preamble for the destination memory.

The DATA query tells the instrument to output the waveform record stored in the waveform memory or channel buffer, previously specified with a :WAVEFORM:SOURCE command, over the HP-IB.

Command Syntax

```
:WAVeform:DATA  <binary block data in # format>
```

Example

```
OUTPUT 707;":WAV:DATA #800001024<block>"
```

Query Syntax

```
:WAVeform:DATA?
```

Returned Format

```
<binary block length bytes><binary block><NL>
```

The following program moves data from the HP 70703A to the controller and then back to the HP 70703A with the :WAVEFORM:DATA query and command.

```
10 CLEAR 707
20                                                       ! Acquire channel 1 data
30 OUTPUT 707;"*CLS*RST"                                 ! Start with known state
40 OUTPUT 707;"ACQUIRE:TYPE NORMAL;COUNT 1;POINTS 512"   ! Setup digitize
50 OUTPUT 707;"DIGITIZE CHANNEL1"                        ! Digitize channel 1
60                                                       ! Setup to transfer data
                                                         to controller
70 OUTPUT 707;"SYSTEM:HEADER OFF"                        ! No headers (compaba-
                                                         bility command)
80 OUTPUT 707;"WAVEFORM:SOURCE CHANNEL1;FORMAT WORD"     ! Waveform data source
                                                         and format
90                                                       ! Transfer waveform
                                                         preamble to controller
100 DIM Preamble$[200]                                   ! Store preamble in this
110 OUTPUT 707;"WAVEFORM:PREAMBLE?"                      ! Query waveform preamble
120 ENTER 707;Preamble$
130                                                      ! Transfer waveform
                                                         data to controller
140 OUTPUT 707;"WAVEFORM:DATA?"                          ! Query waveform data
150 ENTER 707 USING "#,1A,1D";Sharp$,Digits              ! Enter '#b' where b is
                                                         number of digits in byte
                                                         count
160 ENTER 707 USING:"#,"&VAL$(Digits)&"D";Bytes          ! Enter correct number
                                                         of digits in byte count
170 Length=Bytes/2                                       ! Length in words is /2
                                                         number of bytes
180 ALLOCATE INTEGER Waveform(1:Length)                  ! Create array of cor-
                                                         rect length
190 ENTER 707 USING "#,W";Waveform(*)                    ! and fill it with the
                                                         waveform
200 ON TIMEOUT 7,1 GOTO Skip_nl                          ! Need timeout in case
                                                         no trailing <NL> sent
210 ENTER 707 USING "-K,B";End$                          ! Read trailing <NL>
220 Skip_nl: OFF TIMOUT 7                                ! Cancel timeout from
                                                         above
230                                                      ! Transfer data back
                                                         to 70703A into a wave-
                                                         form memory
240 OUTPUT 707;"WAVEFORM:SOURCE WMEMORY3"                ! Store in waveform
                                                         memory 3
250 OUTPUT 707;"WAVEFORM:PREAMBLE ";Preamble$            ! Send preamble
```

```
260 OUTPUT 707;"WAVEFORM:DATA #";            ! Send waveform data command
270 OUTPUT 707;VAL$(LEN(VAL$(Bytes)));       ! with number of digits in byte count
280 OUTPUT 707;VAL$(Bytes);                  ! followed by byte count
290 OUTPUT 707 USING "W";Waveform(*)         ! and waveform data terminated with <NL>
300                                          ! Display data
310 OUTPUT 707;"DISPLAY:FORMAT 2"            ! with channel in upper area
320 OUTPUT 707;"VIEW WMEMORY3"               ! and waveform memory in lower
330 !
340 LOCAL 707
350 !
360 END
```

Note In program line 250, the space after :WAVEFORM:PREAMBLE and before the quote mark is necessary.

FORMat

The :WAVEFORM:FORMAT command sets the data transmission mode for waveform data output. This command controls how the data is formatted on the HP-IB or MSIB when sent from the HP 70703A.

WORD formatted data transfers as 16-bit binary integers in two bytes, with the most significant byte of each word sent first.

BYTE and COMPRESSED formatted data is transferred as 8-bit bytes.

The FORMAT query returns the current output format for transfer of waveform data.

Command Syntax

```
:WAVeform:FORMat {WORD|BYTE|COMPressed}
```

Example

```
OUTPUT 707;":WAV:FORMAT WORD"
```

Query Syntax

```
:WAVeform:FORMat?
```

Returned Format

```
<mode><NL>
```

Where:

```
<mode>::= {WORD|BYTE|COMPressed}
```

Example

```
DIM Frmt$[30]
OUTPUT 707;":WAV:FORMAT?"
ENTER 707;Frmt$
PRINT Frmt$
```

POINts

The :WAVEFORM:POINTS query returns the points value in the currently selected waveform preamble. The points value is the number of time buckets contained in the waveform selected with the :WAVEFORM:SOURCE command.

In most cases the number of time buckets actually acquired will be the number of points set in the ACQUIRE subsystem. There are some sweep speeds where the actual number of points will be less than requested. These are shown below.

With the sweep speed set to 200 ps per division, the number of points actually acquired will be 32, 64, or 100.

With the sweep speed set to 2 ns per division, the number of points actually acquired will be 32, 64, 128, or 200.

With the sweep speed set to 5 ns per division, the number of points actually acquired will be 32, 64, 128, 256, or 500.

With the sweep speed set to 10 ns per division, the number of points actually acquired will be 32, 64, 128, 256, 500, 512, or 1000.

Query Syntax

```
:WAVeform:POINts?
```

Returned Format

```
<value><NL>
```

Where:

```
<value> ::= number of acquired data points (integer-NR1 format)
```

Example

```
Dim Pts$[50]
OUTPUT 707;":WAV:POINTS?"
ENTER 707;Pts$
PRINT Pts$
```

PREamble

The :WAVEFORM:PREAMBLE command sends a waveform preamble to the previously selected waveform memory in the instrument.

The PREAMBLE query sends a waveform preamble to the controller from the waveform source.

Command Syntax

```
:WAVeform:PREamble <preamble block>
```

Where:

```
<preamble block> ::= <format NR1>,<type NR1>,<points NR1>,<count NR1>,
                     <xincrement NR3>,<xorigin NR3>,<xreference NR3>,
                     <yincrement NR3>,<yorigin NR3>,<yreference NR3>
```

Query Syntax

```
:WAVeform:PREamble?
```

Returned Format

```
<preamble block><NL>
```

Where:

```
<preamble block> ::= <format NR1>,<type NR1>,<points NR1>,
                        <count NR1>,<xincrement NR3>,<xorigin NR3>,
                        <xreference NR1>,<yincrement NR3>,<yorigin NR3>,
                        <yreference NR1>
```

Where:

```
<format> ::= 1 for BYTE format
             2 for WORD format
             4 for COMPRESSED format

<type> ::=   1 for NORMAL type
             2 for AVERAGE type
             3 for ENVELOPE type
```

Example

This example program uses both the command and query form of the PREAMBLE command. First the preamble is queried (output to the controller). Then the preamble is returned to the previously selected waveform memory.

```
10  DIM Pre$[120]
20  OUTPUT 707;"SYSTEM:HEADER OFF"
30  OUTPUT 707;"WAVEFORM:PREAMBLE?"
40  ENTER 707 USING "-K";Pre$
50  OUTPUT 707 USING "#,K";"WAV:PREAMBLE ";Pre$
60  END
```

Note In line 50 of the program example, a space is inserted between the word "PREAMBLE" and the closed quotation mark. This space must be inside the quotation mark because in this format (#,K) the data is packed together. Failure to add the space will produce a word that is not a proper command word.

Example

The following program example brings the preamble in a numeric array.

```
10  DIM Preamble[1:10]
20  OUTPUT 707;"SYSTEM:HEADER OFF"
30  OUTPUT 707;"WAVEFORM:PREAMBLE?"
40  ENTER 707 ;Preamble(*)
50  OUTPUT 707;"WAV:PREAMBLE ";Preamble(*)
60  END
```

SOURce

The :WAVEFORM:SOURCE command selects the channel, waveform memory, or function to be used as the source for the waveform commands.

The SOURCE query returns the currently selected source for the waveform commands.

Command Syntax

```
:WAVeform:SOURce {CHANnel{1|2|3|4}|WMEMory{1|2|3|4}|FUNCtion{1|2}}
```

Example

```
OUTPUT 707;":WAV:SOURCE WMEMORY3"
```

Query Syntax

```
:WAVeform:SOURce?
```

Returned Format

```
<source><NL>
```

Where:

```
<source> ::= {CHANnel{1|2|3|4}|WMEMory{1|2|3|4}|FUNCtion{1|2}}
```

Example

```
DIM Src$[30]
OUTPUT 707;":WAVEFORM:SOURCE?"
ENTER 707;Src$
PRINT Src$
```

TYPE

The :WAVEFORM:TYPE query returns the data type for the previously specified waveform source.

Query Syntax

```
:WAVeform:TYPE?
```

Returned Format

```
<mode><NL>
```

Where:

```
<mode> ::= {AVERage|ENVelope|NORMal}
```

Example

```
DIM Typ$[30]
OUTPUT 707;":WAVEFORM:TYPE?"
ENTER 707;Typ$
PRINT Typ$
```

XINCrement

The :WAVEFORM:XINCREMENT query returns the x-increment value currently in the preamble for the current specified source. This value is the time difference between consecutive data points for NORMAL, AVERAGE, or ENVELOPE data.

Query Syntax

```
:WAVeform:XINCrement?
```

Returned Format

```
<value><NL>
```

Where:

```
<value> ::= x-increment in the current preamble (exponential - NR3 format)
```

Example

```
DIM Xin$[50]
OUTPUT 707;":WAV:XINCREMENT?"
ENTER 707;Xin$
PRINT Xin$
```

XORigin

The :WAVEFORM:XORIGIN query returns the x-origin value currently in the preamble for the current specified source. This value is the time of the first data point in the memory with respect to the trigger point.

Query Syntax

```
:WAVeform:XORigin?
```

Returned Format

```
<value><NL>
```

Where:

```
<value> ::= x-origin value currently in preamble (exponential - NR3 format)
```

Example

```
DIM Xr$[50]
OUTPUT 707;":WAV:XORIGIN?"
ENTER 707;Xr$
PRINT Xr$
```

XREFerence

The :WAVEFORM:XREFERENCE query returns the current x-reference value in the preamble for the current specified source. This value specifies the data point associated with the x-origin data value. For the HP 70703A this value is always zero.

Query Syntax

```
:WAVeform:XREFerence?
```

Returned Format

```
<value><NL>
```

Where:

```
<value> ::= x-reference value in the current preamble, always 0
            (integer - NR1 format)
```

Example

```
DIM Xrf$[50]
OUTPUT 707;":WAV:XREFERENCE?"
ENTER 707;Xrf$
PRINT Xrf$
```

YINCrement

The :WAVEFORM:YINCREMENT query returns the y-increment value currently in the preamble for the current specified source. This value is the voltage difference between consecutive data points.

Query Syntax

```
:WAVeform:YINCrement?
```

Returned Format

```
<value><NL>
```

Where:

```
<value> ::= y-increment value in the current preamble
            (exponential - NR3 format)
```

Example

```
DIM Yin$[50]
OUTPUT 707;":WAV:YINCREMENT?"
ENTER 707;Yin$
PRINT Yin$
```

YORigin

The :WAVEFORM:YORIGIN query returns the y-origin currently in the preamble for the current specified source. This value is the voltage at center range.

Query Syntax

```
:WAVeform:YORigin?
```

Returned Format

```
<value><NL>
```

Where:

```
<value> ::= y-origin in the current preamble (exponential - NR3 format)
```

Example

```
Dim Yr$[50]
OUTPUT 707;":WAV:YOR?"
ENTER 707;Yr$
PRINT Yr$
```

YREFerence

The :WAVEFORM:YREFERENCE query returns the current y-reference value in the preamble for the current specified source. This value specifies the data point where the y-origin occurs.

Query Syntax

```
:WAVeform:YREFerence?
```

Returned Format

```
<value><NL>
```

Where:

```
<value> ::= y-reference value in the current preamble
            (integer - NR1 format)
```

Example

```
DIM Yrf$[50]
OUTPUT 707;"WAV:YREFERENCE?"
ENTER 707;Yrf$
PRINT Yrf$
```

A

Algorithms

One of the HP 70703A's primary features is its ability to make automatic measurements on acquired waveforms. This appendix provides details on how automatic measurements are calculated and offers some tips on how to improve results.

Measurement Setup

Measurements typically should be made at the fastest possible sweep speed for the most accurate measurement results. The entire portion of the waveform that is to be measured must be setup and present on the oscilloscope. That is:

- at least one complete cycle must be present for period or frequency measurements.
- the entire pulse must be present for width measurements.
- the leading (positive going) edge of the waveform must be present for risetime measurements.
- the trailing (negative going) edge of the waveform must be present for falltime measurements.

Making Measurements

If more than one waveform, edge, or pulse is present, the measurements are made on the first (leftmost) portion of the active waveform that can be used. If there are not enough data points the HP 70703A will return $\leq$ with the measurement results. This is to remind you that the results may not be as accurate as possible. It is recommended that you rescale the active waveform and make your measurement again.

Standard Measurements

When any of the standard measurements are requested, the HP 70703A first determines the top-base voltage levels at 100%-0%. From this information, it can determine the other important voltage values (10%, 90%, and 50%) needed to make the measurements. The 10% and 90% voltage values are used in the risetime and falltime measurements as well as in all other edge measurements. The 10% and 90% values are also used to determine the 50% value. The 50% voltage value is used for measuring frequency, period, pulse width, and duty cycle.

User-Defined Measurements

The voltage thresholds are precise settings and set specific locations of the waveforem. If the thresholds are not placed on the waveform (above or below) the HP 70703A cannot make a measurement.

When any of the user defined measurements are requested the HP 70703A still must determine the top-base voltage thresholds. From this information it can determine user defined upper and lower thresholds. The midpoint is then determined to be the 50% point between the upper and lower threshold.

Automatic Top-Base

Top-Base is the heart of most automatic measurements. It is used to determine Vtop and Vbase, the 0% and 100% voltage levels at the top and the bottom of the waveform. From this information the oscilloscope can determine the 10%, 50%, and 90% points, which are also used in most measurements. The top or base of the waveform is not necessarily the maximum or minimum voltage present on the waveform. Consider a pulse that has slight overshoot. It would be wrong to select the highest point of the waveform as the top since the waveform normally rests below the perturbation.

Top-Base performs a histogram on the waveform and finds the most prevalent point above and below the waveform midpoint. The most prevalent point is one that represents greater than approximately 5% of the total display points (501) and is considered to be either the top or base. If no point accounts for more than 5% of the total, then the top is chosen as the absolute maximum and the base is chosen as the absolute minimum.

Edge Definition

Both rising and falling edges are defined as transitional edges that must cross three thresholds.

A rising edge must cross the lower threshold in a positive direction (defining it as a rising edge), cross the mid threshold (any number of crossings, both positive and negative are permissible) and then cross the upper threshold without any crossing of the lower threshold.

A falling edge must cross the upper threshold in a negative direction, cross the mid threshold (any number of times), and then cross the lower threshold without crossing the upper threshold.

Note Most time measurements are made based on the position of the first crossing of the middle threshold.

Algorithm Definitions

Following are the definitions that all measurements are based on:

delay

There are three types of delay measurement:

- jitter
- standard
- user-defined

Jitter occurs only under the following circumstances:

- :MEASURE:DELAY is set to STANDARD
- two delay parameters are the same
- :ACQUIRE:TYPE is set to ENVELOPE

```
if

first edge on minimum waveform is rising

then

delay = mid-threshold of first rising edge of max waveform minus
mid-threshold of first rising edge on min waveform

else

delay = mid-threshold of first falling edge on min waveform minus
mid-threshold of first falling edge on max waveform
```

The standard delay measurement occurs when in the standard mode (not user-defined) and is not a jitter measurement.

```
standard delay = mid-threshold of the first edge of second
parameter minus mid-threshold of the first edge of the first parameter
```

Note Negative delay is possible as well as a positive delay.

```
User defined delay = second channel edge minus first channel edge
```

+ width

The + width algorithm has standard and user-defined considerations.

```
if

first edge is rising

then

+ width = mid-threshold crossing of first falling edge - mid-threshold
crossing of first rising edge

else
```

```
+ width = mid-threshold crossing of second falling edge - mid-threshold
crossing of first rising edge
```

User-defined is the same as Standard definition except user-defined threshold.

– width

The – width algorithm has standard and user-defined considerations:

```
if
first edge is rising
then
– width = second rising edge - first falling edge
else
– width = first rising edge - first falling edge
```

Period

```
if
first edge is rising
then
period = second rising edge - first rising edge
else
period = second falling edge - first falling edge
```

Frequency

```
frequency = 1/period
```

Duty Cycle

```
duty cycle = (+ width/period) * 100
```

Note + width is always calculated using mid-threshold.

Risetime

```
risetime = time at upper threshold - time at lower threshold
```

Overshoot

```
if the first edge of the waveform is rising

then

overshoot = (Vmax - Vtop) / Vamplitude

or

overshoot = (Vbase - Vmin) / Vamplitude
```

Preshoot

```
if the first edge of the waveform is rising

then

preshoot = (Vbase - Vmin) / Vamplitude

or

preshoot = (Vmax - Vtop) / Vamplitude
```

Falltime

```
falltime = time at lower threshold - time at upper threshold
```

Vmax

```
Vmax = voltage of the maximum point on screen
```

Vmin

```
Vmin = voltage of the minimum point on screen
```

Vp-p

```
Vp-p = Vmax - Vmin
```

Vtop

Vtop = most prevalent point above waveform midpoint

Vbase

Vbase = most prevalent point below waveform midpoint

Vamp

Vamp = Vtop - Vbase

Vavg

Average voltage of the first cycle of the active signal is measured. If a complete cycle is not present the HP 70703A will average all data points.

Vrms

The rms voltage of the first cycle of the active signal is measured. If a complete cycle is not present, the measurement will compute rms on all data points.

$$V_{rms}(ac) = \left\{ \frac{1}{n} \sum_{j=1}^{n} V_j^2 - \left(\frac{1}{n} \sum_{j=1}^{n} V_j \right)^2 \right\}^{\frac{1}{2}}$$

$$V_{rms}(dc) = \left\{ \frac{1}{n} \sum_{\mathrm{j}=1}^{\mathrm{n}} V_j^2 \right\}^{\frac{1}{2}}$$

B

Message Communication and System Functions

This appendix describes the operation of instruments that operate in compliance with the IEEE 488.2 standard. Instruments that are compatible with IEEE 488.2 must also be compatible with IEEE 488.1, however IEEE 488.1 compatible instruments may or may not conform to the IEEE 488.2 standard. The IEEE 488.2 standard defines the message exchange protocols by which the instrument and the controller will communicate. It also defines some common capabilities, which are found in all IEEE 488.2 instruments. This appendix also contains a few items which are not specifically defined by IEEE 488.2, but deal with message communication or system functions.

Protocols

The protocols of IEEE 488.2 define the overall scheme used by the controller and the instrument to communicate. This includes defining when it is appropriate for devices to talk or listen, and what happens when the protocol is not followed.

Functional Elements

Before proceeding with the description of the protocol, a few system components should be understood.

Input Buffer

The input buffer of the instrument is the memory area where commands and queries are stored prior to being parsed and executed. It allows a controller to send a string of commands to the instrument which could take some time to execute, and then proceed to talk to another instrument while the first is parsing and executing commands. The HP 70703A's input buffer will hold 1024 characters, or bytes of data.

Output Queue

The output queue of the instrument is the memory area where all output data (<response messages>) are stored until read by the controller.

Parser

The instrument's parser is the component that interprets the commands sent to the instrument and decides what actions should be taken. "Parsing" refers to the action taken by the parser to achieve this goal. Parsing and executing of commands begins when either the instrument sees a <program message terminator> (defined later in this appendix) or the input buffer becomes full. If you wish to send a long sequence of commands to be executed and then talk to another instrument while they are executing, you should send all the commands before sending the <program message terminator>.

Protocol Overview

The instrument and controller communicate using <program message>s and <response message>s. These messages serve as the containers into which sets of program commands or instrument responses are placed. <program message>s are sent by the controller to the instrument, and <response message>s are sent from the instrument to the controller in response to a query message. A "query message" is defined as being a <program message> which contains one or more queries. The instrument will only talk when it has received a valid query message, and therefore has something to say. The controller should only attempt to read a response after sending a complete query message, but before sending another <program message>. The basic rule to remember is that the instrument will only talk when prompted to, and it then expects to talk before being told to do something else.

Protocol Operation

When the instrument is turned on or when it receives a device clear command, the input buffer and output queue are cleared, and the parser is reset to the root level of the command tree.

The instrument and the controller communicate by exchanging complete <program message>s and <response message>s. This means that the controller should always terminate a <program message> before attempting to read a response.

If a query message is sent, the next message passing over the bus should be the <response message>. The controller should always read the complete <response message> associated with a query message before sending another <program message> to the same instrument.

The instrument allows the controller to send multiple queries in one query message. This is referred to as sending a "compound query." As will be noted later in this appendix, multiple queries in a query message are separated by semicolons. The responses to each of the queries in a compound query will also be separated by semicolons.

Commands are executed in the order they are received. This also applies to the reception of the group execute trigger (GET) bus command. The group execute trigger command should not be sent in the middle of a <program message>.

Protocol Exceptions

If an error occurs during the information exchange, the exchange may not be completed in a normal manner. Some of the protocol exceptions are shown below.

Addressed to talk with nothing to say. If the instrument is addressed to talk before it receives a query, it will indicate a query error and will not send any bytes over the bus. If the instrument has nothing to say because queries requested were unable to be executed because of some error, the device will not indicate a query error, but will simply wait to receive the next message from the controller.

Addressed to talk with no listeners on the bus. If the instrument is addressed to talk and there are no listeners on the bus, the instrument will wait for a listener to listen, or for the controller to take control.

Command Error

A command error will be reported if the instrument detects a syntax error or an unrecognized command header.

Execution Error

An execution error will be reported if a parameter is found to be out of range, or if the current settings do not allow execution of a requested command or query.

Device-specific Error

A device-specific error will be reported if the instrument is unable to execute a command for a strictly device dependent reason.

Query Error

A query error will be reported if the proper protocol for reading a query is not followed. This includes the interrupted and unterminated conditions described below.

Unterminated Condition

If the controller attempts to read a <response message> before terminating the <program message>, a query error will be generated. The parser will reset itself, and the response will be cleared from the output queue of the instrument without being sent over the bus.

Interrupted Condition

If the controller does not read the entire <response message> generated by a query message and then attempts to send another <program message>, the device will generate a query error. The unread portion of the response will then be discarded by the instrument. The interrupting <program message> will not be affected.

Buffer Deadlock

The instrument may become deadlocked if the input buffer and output queue both become full. This condition can occur if a very long <program message> is sent containing queries that generate a great deal of response data. The instrument cannot accept any more bytes, and the controller cannot read any of the response data until it has completed sending the entire <program message>. Under this condition the instrument will break the deadlock by clearing the output queue, and continuing to discard responses until it comes to the end of the current <program message>. The query error bit will also be set.

Syntax Diagrams

The syntax diagrams in this appendix are similar to the syntax diagrams in the IEEE 488.2 specification. Commands and queries are sent to the instrument as a sequence of data bytes. The allowable byte sequence for each functional element is defined by the syntax diagram that is shown with the element description.

The allowable byte sequence can be determined by following a path in the syntax diagram. The proper path through the syntax diagram is any path that follows the direction of the arrows. If there is a path around an element, that element is optional. If there is a path from right to left around one or more elements, that element or those elements may be repeated as many times as desired.

Syntax Overview

This overview is intended to give a quick glance at the syntax defined by IEEE 488.2. It should allow you to understand many of the things about the syntax you need to know. This appendix also contains the details of the IEEE 488.2 defined syntax.

IEEE 488.2 defines the blocks used to build messages which are sent to the instrument. A whole string of commands can therefore be broken up into individual components.

Figure B-1 shows a breakdown of an example <program message>. There are a few key items to notice:

1. A semicolon separates commands from one another. Each <program message unit> serves as a container for one command. The <program message unit>s are separated by a semicolon.
2. A <program message> is terminated by a <NL> (new line), a <NL> with EOI asserted, or EOI being asserted on the last byte of the message. The recognition of the <program message terminator>, or <PMT>, by the parser serves as a signal for the parser to begin execution of commands. The <PMT> also affects command tree traversal (see the Programming and Documentation Conventions chapter).
3. Multiple data parameters are separated by a comma.
4. The first data parameter is separated from the header with one or more spaces.
5. The header MEAS:SOURCE is a compound header. It places the parser in the measure subsystem until the <NL> is encountered.

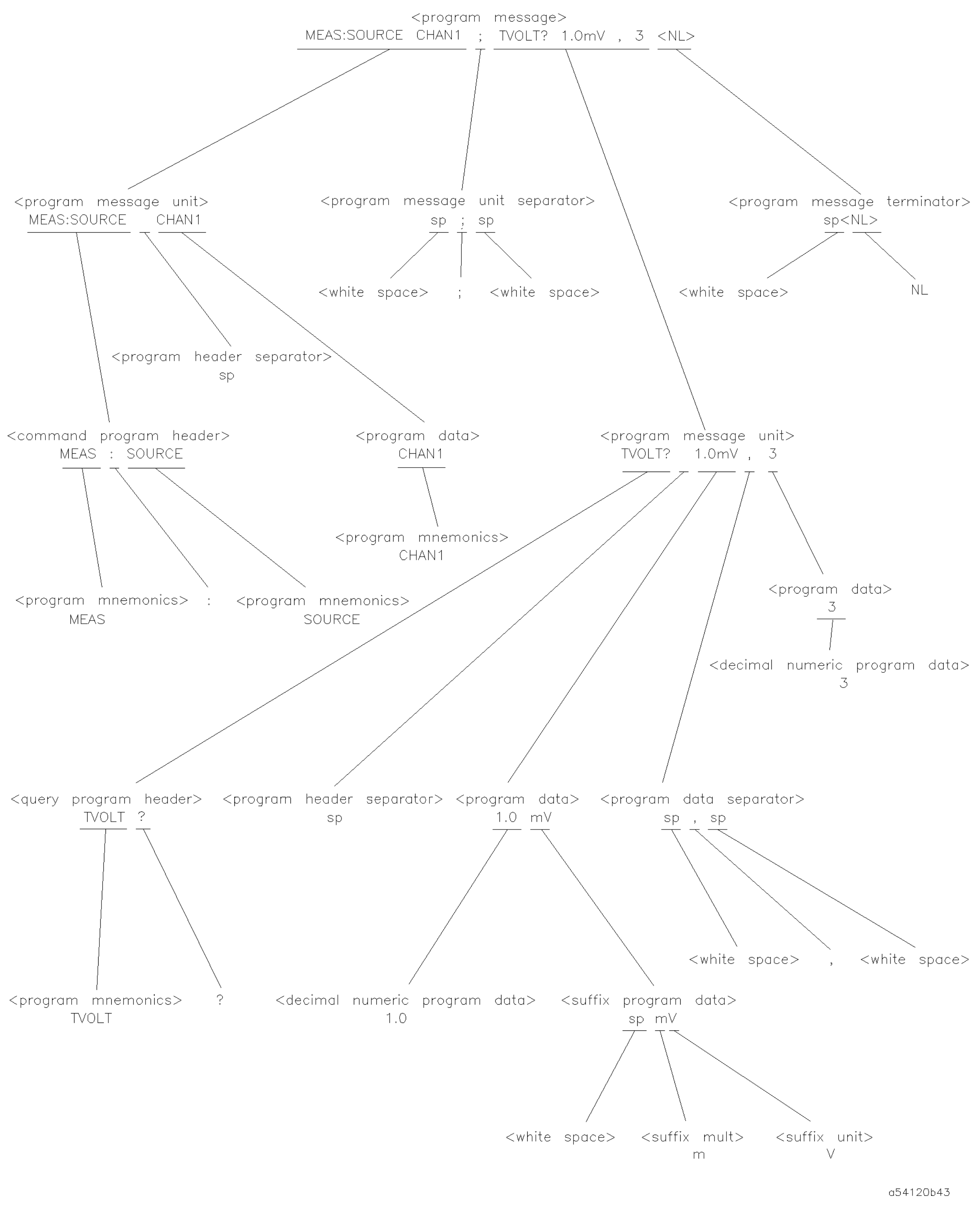


Figure B-1. <program message> Parse Tree

Device Listening Syntax

The listening syntax of IEEE 488.2 is designed to be more forgiving than the talking syntax. This allows greater flexibility in writing programs, as well as allowing them to be easier to read.

Upper/Lower Case Equivalence

Upper and lower case letters are equivalent. The mnemonic RANGE has the same semantic meaning as the mnemonic range.

<white space>

<white space> is defined to be one or more characters from the ASCII set of 0 - 32 decimal, excluding 10 decimal (NL). <white space> is used by several instrument listening components of the syntax.

It is usually optional, and can be used to increase the readability of a program.

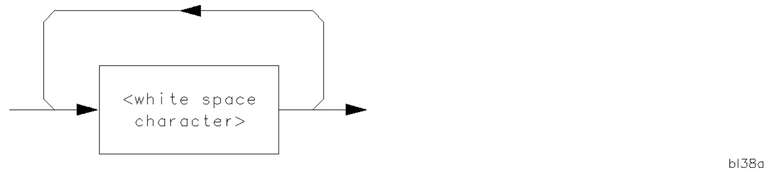


Figure B-2. <white space>

<program message>

The <program message> is a complete message to be sent to the instrument. The instrument will begin executing commands once it has a complete <program message>, or when the input buffer becomes full. The parser is also repositioned to the root of the command tree after executing a complete <program message>. Refer to the Tree Traversal Rules in the Programming and Documentation Conventions chapter for more details.

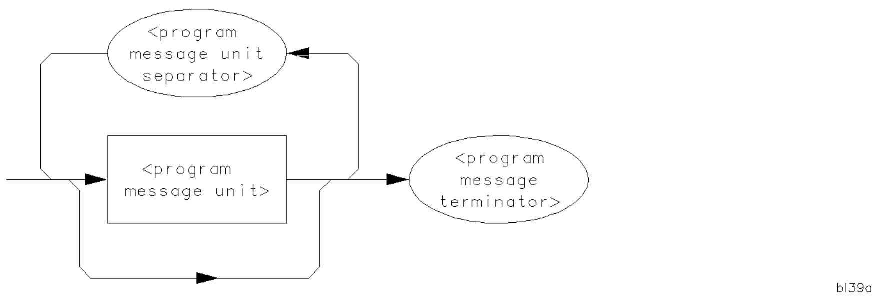


Figure B-3. <program message>

<program message unit>

The <program message unit> is the container for individual commands within a <program message>.

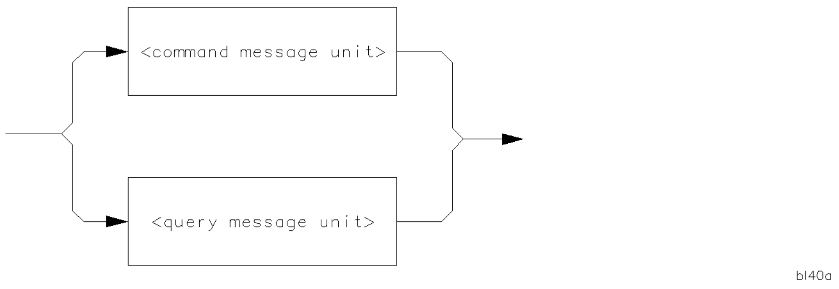


Figure B-4. <program message unit>

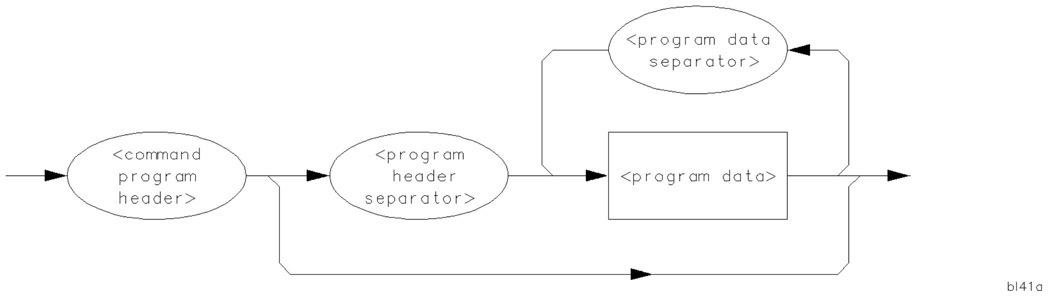


Figure B-5. <command message unit>

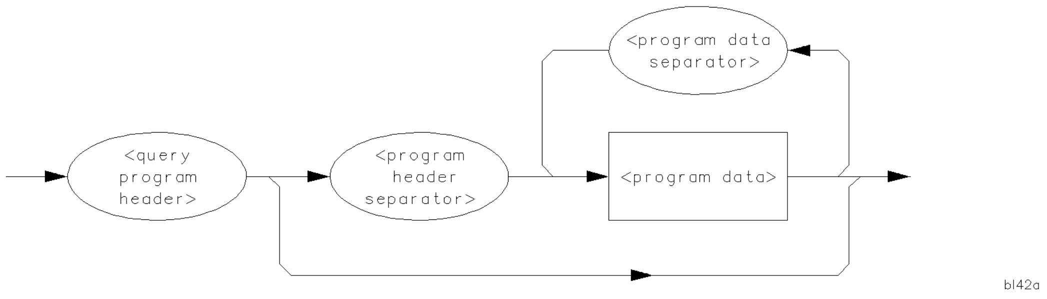


Figure B-6. <query message unit>

<program message unit separator>.

A semicolon separates <program message unit>s, or individual commands.

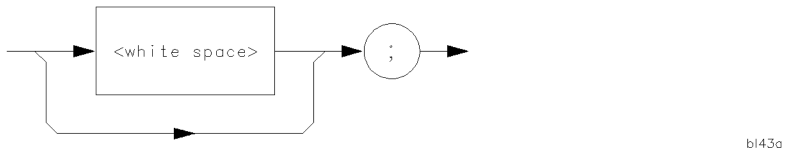
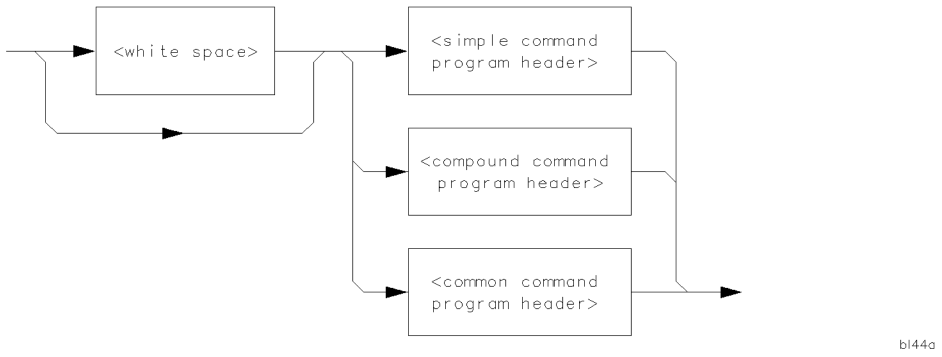


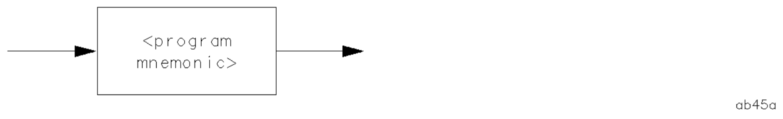
Figure B-7. <program message unit separator>

<command program header>/<query program header>

These elements serve as the headers of commands or queries. They represent the action to be taken.



Where <simple command program header> is defined as



Where <compound command program header> is defined as

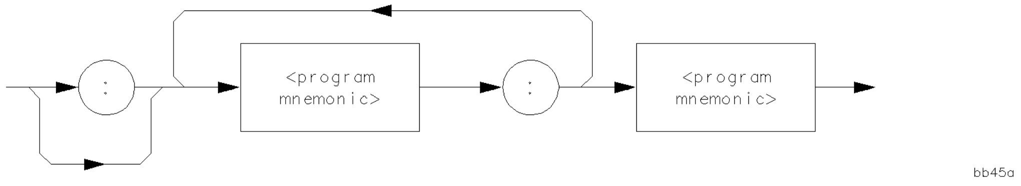
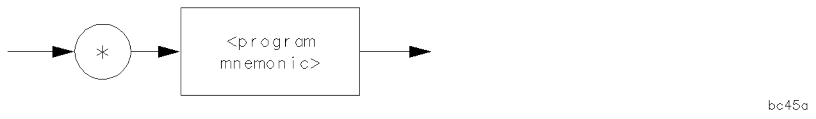


Figure B-8. <command program header>

Where <common command program header> is defined as



Where <program mnemonic> is defined as

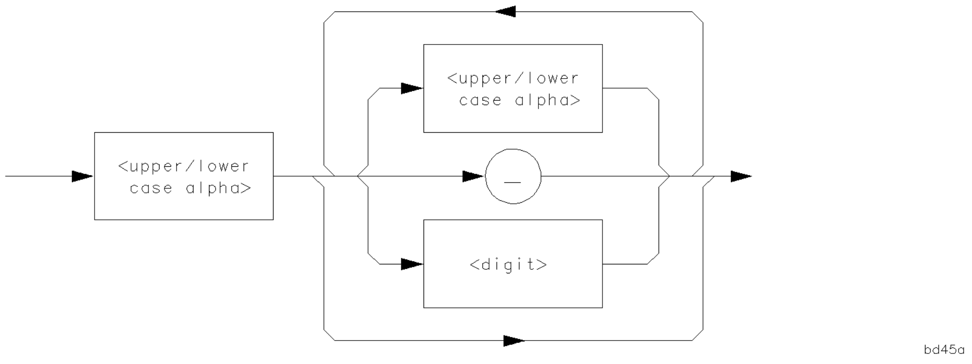
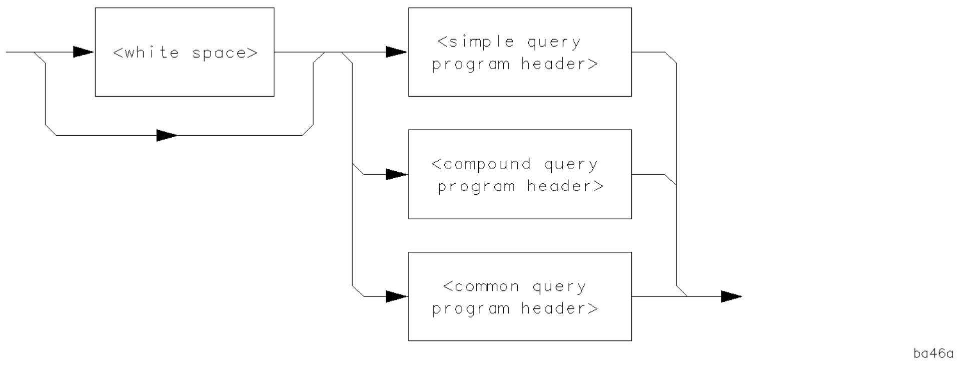


Figure B-8. <command program header> continued

Where <upper/lower case alpha> is defined as a single ASCII encoded byte in the range 41 - 5A, 61 - 7A (65 - 90, 97 - 122 decimal).

Where <digit> is defined as a single ASCII encoded byte in the range 30 - 39 (48 - 57 decimal).

Where (_) represents an <169>underscore<170>, a single ASCII-encoded byte with the value 5F (95 decimal).



Where <simple query program header> is defined as

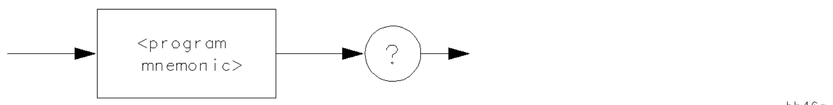
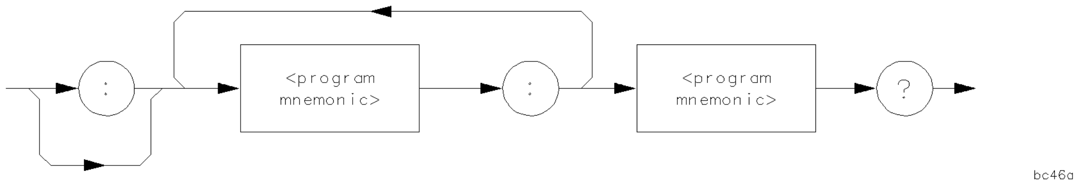


Figure B-9. <query program header>

Where <compound query program header> is defined as



Where <common query program header> is defined as

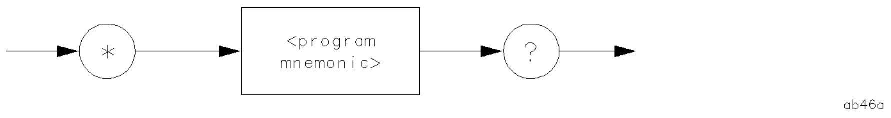


Figure B-9. <query program header> (continued)

<program data>

The <program data> element represents the possible types of data which may be sent to the instrument. The HP 70703A will accept the following data types: <character program data>, <decimal numeric program data>, <suffix program data>, <string program data>, and <arbitrary block program data>.

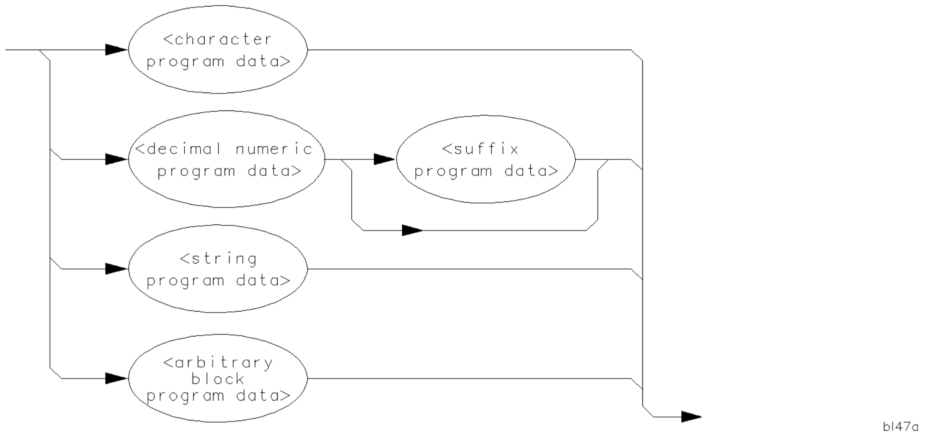


Figure B-10. <program data>

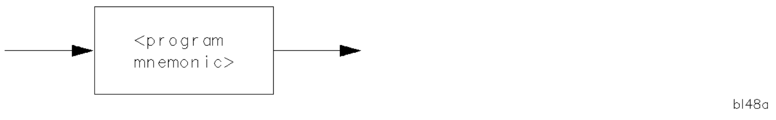
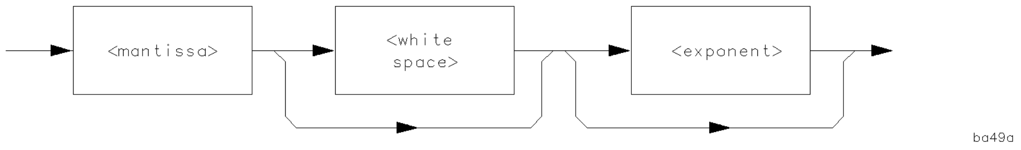
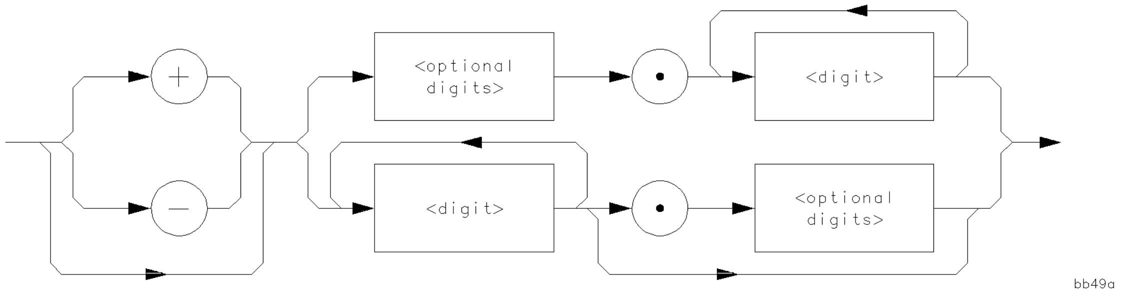


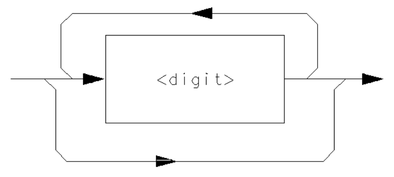
Figure B-11. <character program data>



Where <mantissa> is defined as



Where <optional digits> is defined as



Where <exponent> is defined as

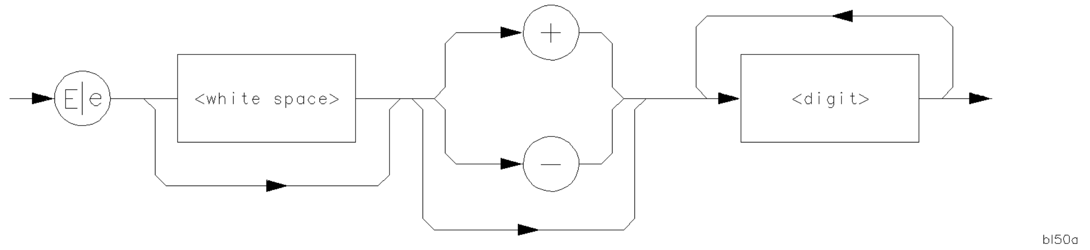


Figure B-12. <decimal numeric program data>

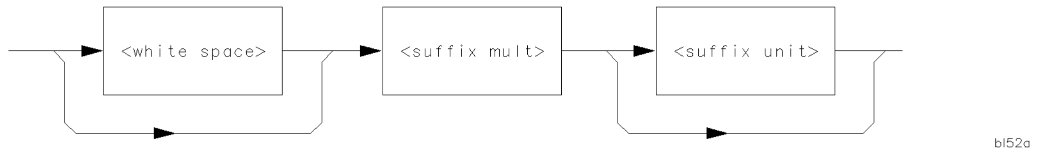


Figure B-13. <suffix program data>

Suffix Multiplier

The suffix multipliers that the instrument will accept are shown in table B-1.

Table B-1. <suffix mult>

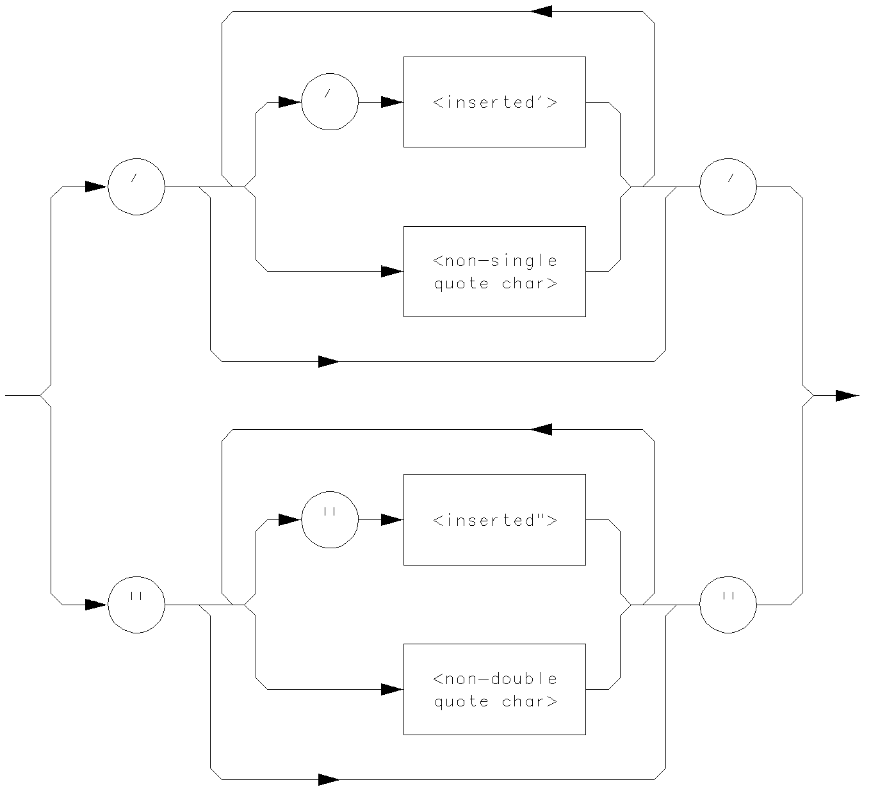
Value	Mnemonic
1E18	EX
1E15	PE
1E12	T
1E9	G
1E6	MA
1E3	K
1E-3	M
1E-6	U
1E-9	N
1E-12	P
1E-15	F
1E-18	A

Suffix Unit

The suffix units that the instrument will accept are shown in table B-2.

Table B-2. <suffix unit>

Suffix	Referenced Unit
V	Volt
S	Second



Where <inserted'> is defined as a single ASCII character with the value 27 (39 decimal).

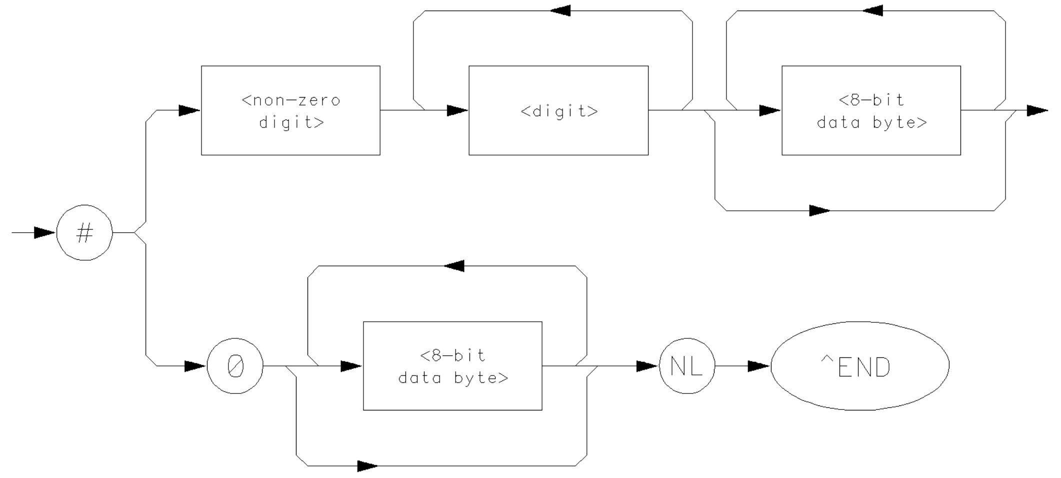
Where <non-single quote char> is defined as a single ASCII character of any value except 27 (39 decimal).

Where <inserted"> is defined as a single ASCII character with the value 22 (34 decimal).

Where <non-double quote char> is defined as a single ASCII character of any value except 22 (34 decimal).

bl53a

Figure B-14. <string program data>



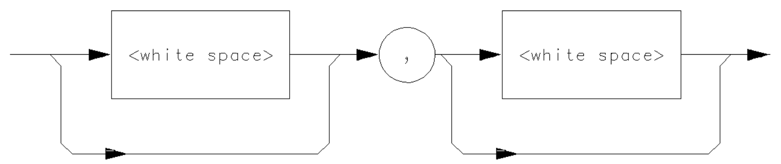
Where <non-zero digit> is defined as a single ASCII encoded byte in the range 31–39 (49–57 decimal).
Where <8-bit byte> is defined as an 8-bit byte in the range 00 –ff (0–255 decinal).

bl54a

Figure B-15. <arbitrary block program data>

<program data separator>

A comma separates multiple data parameters of a command from one another.

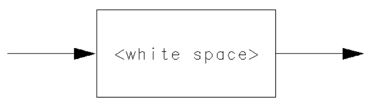


bl55a

Figure B-16. <program data separator>

<program header separator>

A space (ASCII decimal 32) separates the header from the first or only parameter of the command.

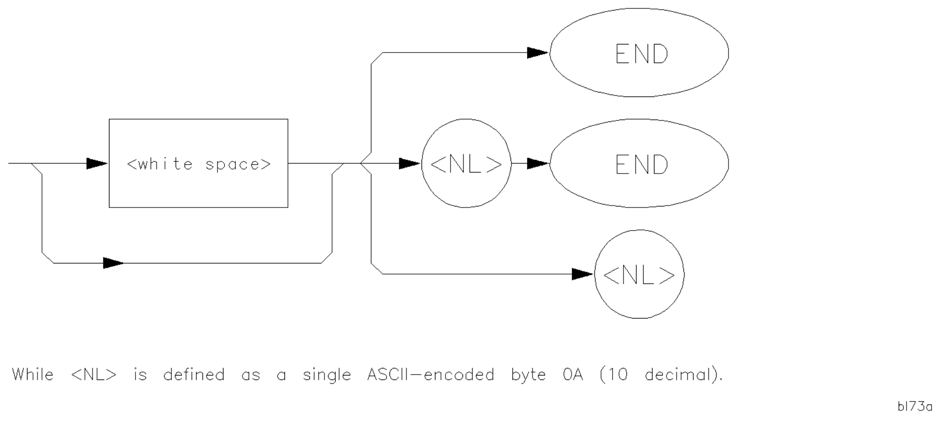


bl56a

Figure B-17. <program header separator>

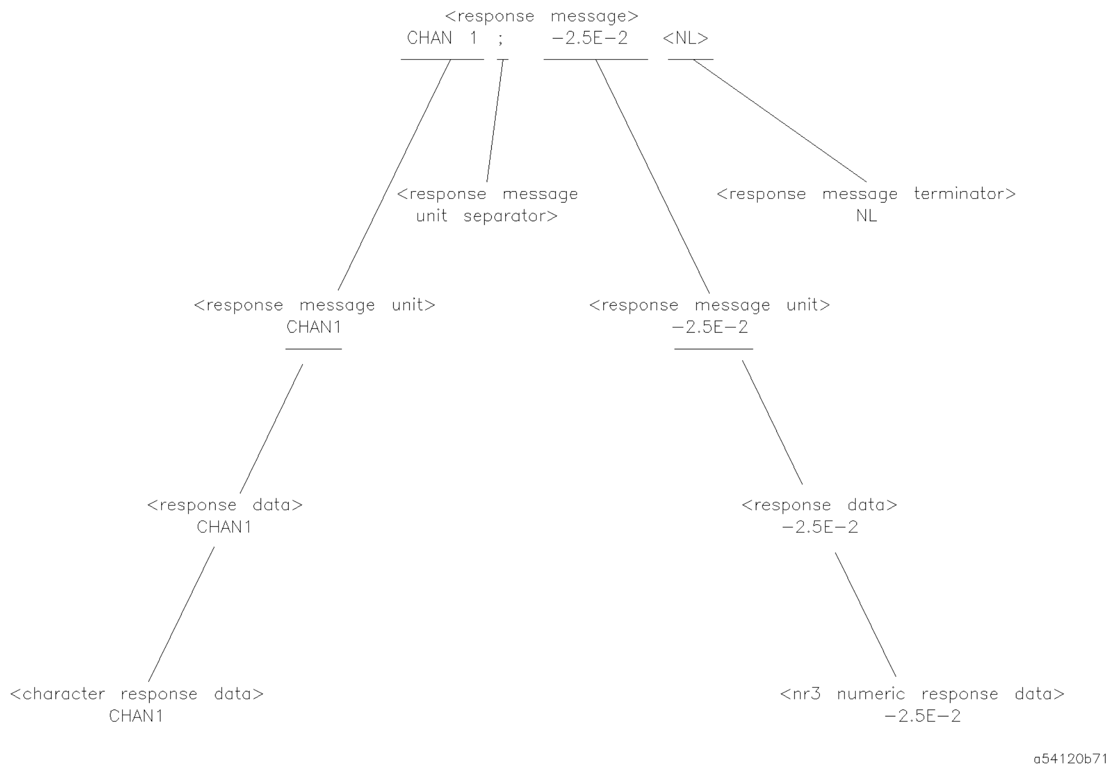
<program message terminator>

The <program message terminator> or <PMT> serves as the terminator to a complete <program message>. When the parser sees a complete <program message> it will begin execution of the commands within that message. The <PMT> also resets the parser to the root of the command tree.



<white space>
<NL>
END
END
<NL>
While <NL> is defined as a single ASCII-encoded byte 0A (10 decimal).
bl73a

Figure B-18. <program message terminator>



<response message>
CHAN 1 ; -2.5E-2 <NL>
<response message unit separator>
<response message terminator> NL
<response message unit> CHAN1
<response message unit> -2.5E-2
<response data> CHAN1
<response data> -2.5E-2
<character response data> CHAN1
<nr3 numeric response data> -2.5E-2
a54120b71

Figure B-19. <response message tree>

Device Talking Syntax

The talking syntax of IEEE 488.2 is designed to be more precise than the listening syntax. This allows the programmer to write routines which can more easily interpret and use the data the instrument is sending. One of the implications of this is the absence of <white space> in the talking formats. The instrument will not pad messages which are being sent to the controller with spaces.

<response message>

This element serves as a complete response from the instrument. It is the result of the instrument executing and buffering the results from a complete <program message>. The complete <response message> should be read before sending another <program message> to the instrument.

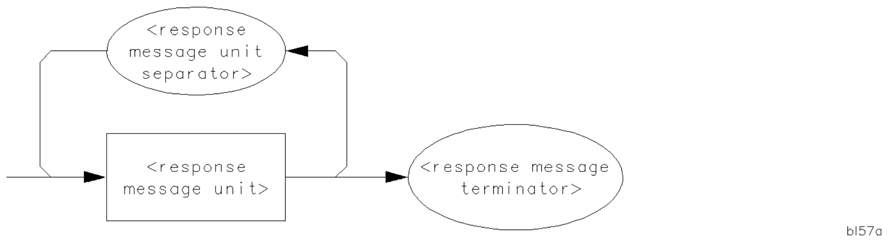


Figure B-20. <response message>

<response message unit>

This element serves as the container of individual pieces of a response. Typically a <query message unit> will generate one <response message unit>, although a <query message unit> may generate multiple <response message unit>s.

A <response message unit> contains one or more <response data> separated by <response data separator>.

<response data>

The <response data> element represents the various types of data which the instrument may return. These types include: <character response data>, <NR1 numeric response data> (integer),<NR3 numeric response data> (exponential), <string response data>, <definite length arbitrary block response data>, and <arbitrary ASCII response data>.

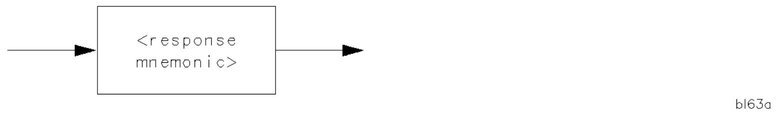


Figure B-21. <character response data>

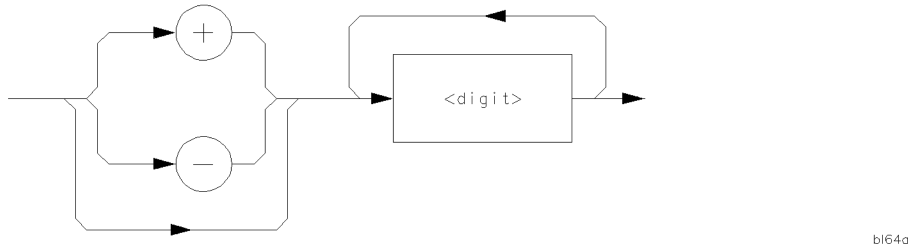


Figure B-22. <NR1 numeric response data>

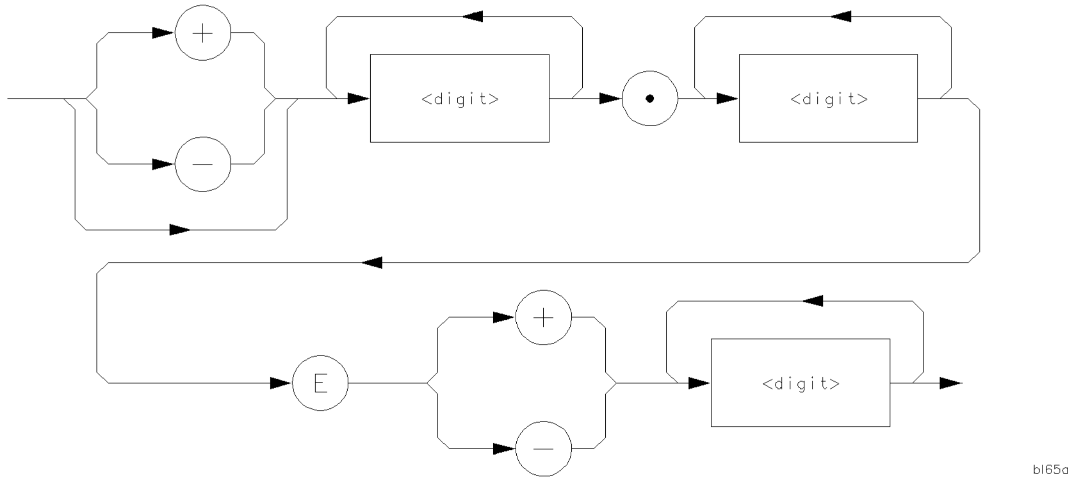


Figure B-23. <NR3 numeric response data>

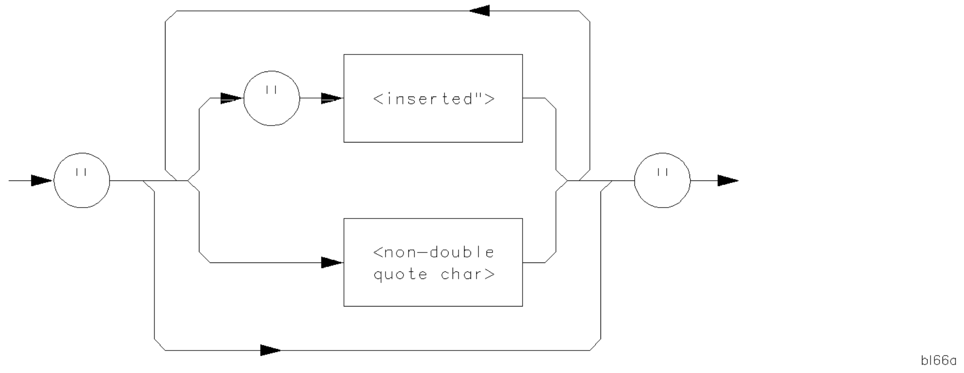


Figure B-24. <string response data>

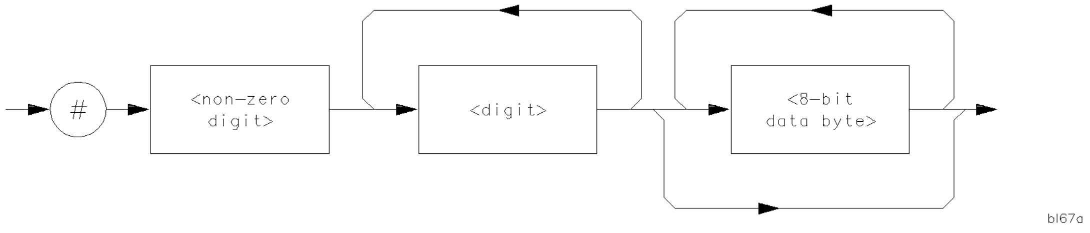


Figure B-25. <definite length arbitrary block>

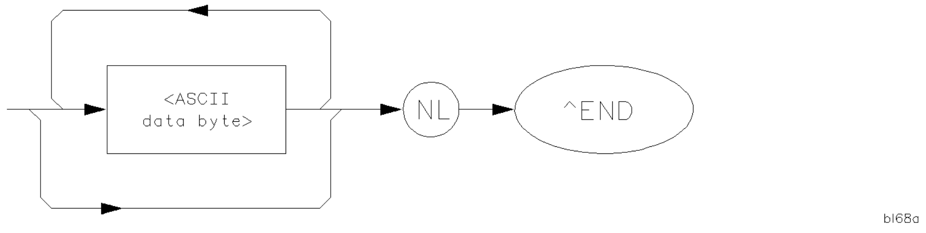


Figure B-26. <arbitrary ASCII response data>

<response data separator>

A comma separates multiple pieces of response data within a single <response message unit>.

Figure B-27. <response data separator>

<response message unit separator>

A semicolon delimits the <response message unit>s if multiple responses are returned.

Figure B-28. <response message unit separator>

<response message terminator>

A <response message terminator> (NL) terminates a complete <response message>. It should be read from the instrument along with the response itself.

Note If you do not read the <response message terminator> the HP 70703A will produce an interrupted error on the next message.

Common Commands

IEEE 488.2 defines a set of common commands. These commands perform functions which are common to any type of instrument. They can therefore be implemented in a standard way across a wide variety of instrumentation. All the common commands of IEEE 488.2 begin with an asterisk. There is one key difference between the IEEE 488.2 common commands and the rest of the commands found in this instrument. The IEEE 488.2 common commands do not affect the parser's position within the command tree. More information about the command tree and tree traversal can be found in the Programming and Documentation Conventions chapter.

Table B-3. HP 70703A/s IEEE 488.2 Common Commands

Command	Command Name
*CLS	Clear Status Command
*ESE	Event Status Enable Command
*ESE?	Event Status Enable Query
*ESR?	Event Status Register Query
*IDN?	Identification Query
*LRN?	Learn Device Setup Query
*OPC	Operation Complete Command
*OPC?	Operation Complete Query
*OPT?	Option Identification Query
*RCL	Recall Command
*RST	Reset Command
*SAV	Save Command
*SRE	Service Request Enable Command
*STB?	Read Status Byte Query
*TRG	Trigger Command
*TST?	Self-Test Query
*WAI	Wait-to-Continue Command

C

Programming Example

Waveform Mask Test Program Using Functions

This program is written to work with the HP 70703A, the HP 54503A and the HP E1426A 500 MHz digitizing oscilloscopes. The program will capture a eference waveform called a “Golden” waveform, then use it as the tandard for generating a template of upper and lower boundaries. The upper and lower boundaries are determined by sliding an ellipse along the reference “Golden” waveform. The ellipse is defined by user specified arameters. Horizontal and vertical tolerances are entered as the dimension of the ellipse. The ellipse is drawn on the controller screen to show the tolerances that will be used to generate the boundary waveforms. Then the tolerance waveforms are drawn on the controller screen. The example is written using:

- an HP-IB select code of 7, and secondary address of 07 for the oscilloscope
- an HP Series 200/300 computer with HP BASIC
- an HP HP 8116A or suitable pulse generator

Execute

```
 10  OPTION BASE 1                          ! Arrays start from 1
 20  !
 30  ! _________________ ASSIGN I/O paths _________________
 40  !
 50  COM /Scope/ INTEGER Scope_sc,INTEGER Scope_ba,@Scope,@Fscope
 60  !
 70  Note: A 54503A must be in addressedmode with EOI on.
 80  !
 90  Scope_sc=7                              ! Select code of HP-IB card
100  Scope_ba=7                              ! HP-IB bus address of scope
110  !
120  ASSIGN @Scope TO Scope_sc*100+Scope_ba   ! Normal transfers
130  ASSIGN @Fscope TO Scope_sc*100+Scope_ba;FORMAT OFF ! Fast transfers
140  !
150  ! _________________ Initialize Scope Variables _________________
160  !
170  INTEGER Gold_wf(1:500)
180  INTEGER Goldmax(1:500)
190  INTEGER Goldmin(1:500)
200  !
210  INTEGER Bytes
220  INTEGER Length
230  INTEGER A
240  !
250  DIM Gold_pre$[200]
260  !
270  Length=500                              ! Use 500pts full scale time
280  !
290  CLEAR SCREEN                            ! Clear alpha screen
300  GCLEAR                                  ! Clear graphics screen
310  GINIT                                   ! Initialize graphics
320  GRAPHICS ON                             ! Turn graphics display on
330  !
340  ! _________________ Display Documentation _________________
350  !
```

```
360  OUTPUT CRT;"This program is written to work with the HP 54503A. It
     will also work with"
370  OUTPUT CRT;"the HP E1426A or HP 70703A using the COMP language."
380  OUTPUT CRT;" "
390  !
400  OUTPUT CRT;"This program will capture a reference waveform  called
     a ""Golden"" waveform."
410  OUTPUT CRT;"It will then use this as a standard for generating a
     template of upper and"
420  OUTPUT CRT;"lower bounds. The upper and lower bounds are determined by
     sliding an elipse"
430  OUTPUT CRT;" along the reference ""Golden"" waveform."
440  OUTPUT CRT;" "
450  !
460  OUTPUT CRT;" The elipse is defined by user specified parameters.
     Horizontal and vertical"
470  OUTPUT CRT;" tolerances are entered as the dimensions of the
     elipse."
480  OUTPUT CRT;" "
490  OUTPUT CRT;" The elipse is drawn on the controller screen to show
     the tolerances that will"
500  OUTPUT CRT;" be used to generate the boundary waveforms. Then the
     tolerance waveforms"
510  OUTPUT CRT;" are shown on the controller screen."
520  OUTPUT CRT;" "
530  !
540  ! __________________ Begin setup for Acquisition __________________
550  !
560  CLEAR @Scope
570  OUTPUT @Scope;"*RST"                     ! Initialize all scope parameters
580  !
590  INPUT "Connect a reference signal to Chan 1 then CONT/RETURN/ENTER" ,A
600  !
610  OUTPUT @Scope;":AUTOSCALE"               ! Find a Golden signal
620  OUTPUT @Scope;":ACQUIRE:complete 100;type AVERAGE ;count 8;points";Length
630  !
640  ! ____________ Acquire (DIG) and get the reference waveform ____________
650  !
660  CALL Get_golden_wf(Gold_wf(*),Bytes,Gold_pre$)
670  MAT Goldmax= Gold_wf                     ! Copy gold waveform into max
680  MAT Goldmin= Gold_wf                     ! Copy gold waveform into min
690  !
700  ! Generate an elipse then use it to establish upper and lower bounds
710  !
720  CLEAR SCREEN
730  CALL Set_tolerance(Length,Gold_wf(*),Goldmin(*),Goldmax(*))
740  !
750  ! __________________ Write to the scope __________________
760  !
770  CALL Put_template(Bytes,Goldmax(*),Goldmin(*),Gold_pre$)
780  CALL Draw_lims(Goldmax(*),Goldmin(*))
790  CALL Test_func (Goldmax(*),Goldmin(*))
800  !
810  OUTPUT @Scope;":ACQUIRE:type NORMAL"
820  !
830  !   AT THIS POINT ONE COULD CHECK THE RESULTS PERIODICALLY WITH "MEAS:RES?"
840  !
850  !   ONE SHOULD SEE TWO RESULTS RETURNED, BOTH VMIN MEASUREMENT, THE FIRST
860  !   IS FOR FUNC1 AND THE SECOND IS FOR FUNC2.
870  !
880  !   BY TWEEKING THE ACTUAL SIGNAL OUTSIDE THE LIMITS, EITHER VMIN FOR FUNC1
```

```
890  !   OR VMIN FOR FUNC2 SHOULD FAIL.
900  !
910  REPEAT
920    WAIT .1                              ! Only check 10 times per second
930    Stb=SPOLL(@Scope)
940  UNTIL (BINAND(Stb,8)<>0)
950  !
960  BEEP
970  PRINT "Limit test failed!"
980  !
990  END
1000 !
1010 !************************END MAIN PROGRAM***********************
1020 !
1030 SUB Get_golden_wf(INTEGER Gold_wf(*),INTEGER Bytes,Gold_pre$)
1040   !
1050   ! This routine will capture data to channel one using the DIG command.
1060   ! Then read the data into Gold_wf(*) along with associated scale info.
1070   ! Use format WORD and read direct into INTEGER array.
1080   !
1090   COM /Scope/ INTEGER Scope_sc,INTEGER Scope_ba,@Scope,@Fscope
1100   !
1110   OUTPUT @Scope;":system:header off" ! Control header off for query reponses
1120   OUTPUT @Scope;":waveform:source channel1;format word"
1130   OUTPUT @Scope;":digitize channel1"     ! Capture data to ch1 and stop
1140   OUTPUT @Scope;":waveform:preamble?"    ! Read an ASCII string of REAL scale factors
1150   ENTER @Scope USING "-K";Gold_pre$
1160   !
1170   OUTPUT @Scope;":waveform:data?"        ! Go read the data
1180   ENTER @Scope USING "#,1A,1D";Header$,Digits
1190   ENTER @Scope USING "#,"&VAL$(Digits)&"D";Bytes
1200   Length=Bytes/2                         ! Normally used to ALLOCATE
1210   ENTER @Fscope;Gold_wf(*)               ! but NOT in this example.
1220   ENTER @Scope USING "-K,B";End$         ! Swallow a linefeed IEEE488.2
1230 SUBEND
1240 !
1250 !--------------------------------------------------------------
1260 !
1270 SUB Put_template(INTEGER Bytes,INTEGER Goldmax(*),INTEGER Goldmin(*),Gold_pre$)
1280   ! This routine will output the boundary waveforms to wmemory1 (high) and
1290   ! wmemory2 (low). Scale information is also written out.
1300   !
1310   COM /Scope/ INTEGER Scope_sc,INTEGER Scope_ba,@Scope,@Fscope
1320   !
1330   OUTPUT @Scope;":waveform:source wmemory2"
1340   OUTPUT @Scope USING "#,K";":waveform:preamble ";Gold_pre$
1350   OUTPUT @Scope USING "#,K,K";":waveform:data #"&VAL$
       (LEN(VAL$(Bytes)));Bytes
1360   OUTPUT @Fscope;Goldmin(*)
1370   OUTPUT @Scope;" "
1380   OUTPUT @Scope;":view wmemory2"
1390   !
1400   OUTPUT @Scope;":waveform:source wmemory1"
1410   OUTPUT @Scope USING "#,K";":waveform:preamble ";Gold_pre$
1420   OUTPUT @Scope USING "#,K,K";":waveform:data#"&VAL$(LEN(VAL$(Bytes)));Bytes
1430   OUTPUT @Fscope;Goldmax(*)
1440   OUTPUT @Scope;" "
1450   OUTPUT @Scope;":view wmemory1"
1460 SUBEND
1470 !
1480 !--------------------------------------------------------------
```

```
1490 !
1500 SUB Test_func(INTEGER Goldmin(*),INTEGER Goldmax(*))
1510   ! Use the functions to do limit testing
1520   !
1530   COM /Scope/ INTEGER Scope_sc,INTEGER Scope_ba,@Scope,@Fscope
1540   !
1550   Mingold=MIN(Goldmin(*))
1560   Maxgold=MAX(Goldmax(*))
1570   !
1580   OUTPUT @Scope;":channel1:range?"
1590   ENTER @Scope;Range
1600   OUTPUT @Scope;":view function1; :view function2"        ! Turn ON
1610   OUTPUT @Scope;":function1:subtract wmemory1, channel1"  ! Define
1620   OUTPUT @Scope;":function1:range ";Range                 ! Scale
1630   OUTPUT @Scope;":function2:subtract channel1, wmemory2"
1640   OUTPUT @Scope;":function2:range ";Range
1650   OUTPUT @Scope;":measure:statistics on"
1660   !
1670   OUTPUT @Scope;":measure:source function1"
1680   OUTPUT @Scope;":measure:vmin"
1690   !
1700   OUTPUT @Scope;":measure:source function2"
1710   OUTPUT @Scope;":measure:vmin"
1720   !
1730   OUTPUT @Scope;":measure:compare vmin, 270,0;postfailure stop"  ! Setup
1740   OUTPUT @Scope;":waveform:source channel1;:measure:destination wmem3"
1750   OUTPUT @Scope;":store chan1,wmem3;:view wmem3"
1760   OUTPUT @Scope;"*sre 8;*cls"
1770   OUTPUT @Scope;":measure:limittest measure;:run"          ! Run test
1780 SUBEND
1790 !
1800 !--------------------------------------------------------------
1810 !
1820 SUB Draw_e(INTEGER Waveform(*),INTEGER A)
1830   ! Routine to draw the tolerance elipse on the controller screen
1840   !
1850   VIEWPORT 30,100*RATIO,15,85
1860   WINDOW -250,250,-32640/2,32640/2
1870   !
1880   FRAME
1890   PEN 3
1900   FOR I=-A TO A
1910     MOVE I,Waveform(I)
1920     DRAW I,Waveform(I)
1930     MOVE I,-Waveform(I)
1940     DRAW I,-Waveform(I)
1950   NEXT I
1960 SUBEND
1970 !
1980 !--------------------------------------------------------------
1990 !
2000 SUB Draw_lims(INTEGER Goldmax(*),INTEGER Goldmin(*))
2010   ! Routine to draw the min and max waveforms on the controller screen
2020   !
2030   VIEWPORT 30,100*RATIO,15,85
2040   WINDOW 1,500,0,32640
2050   !
2060   PEN 5
2070   FOR I=1 TO 500
2080     MOVE I,Goldmax(I)
2090     DRAW I,Goldmax(I)
```

```
2100    MOVE I,Goldmin(I)
2110    DRAW I,Goldmin(I)
2120  NEXT I
2130 SUBEND
2140 !
2150 !---------------------------------------------------------------
2160 !
2170 SUB Set_tolerance(INTEGER Length,INTEGER Gold_wf(*),INTEGER Goldmin(*),INTEGER Goldmax(*))
2180  ! Input: Gold_wf(*) is the reference waveform
2190  !       Length is the record length (500)
2200  ! Output: Goldmin(*) is the min boundary waveform
2210  !         Goldmax(*) is the max boundary waveform
2220  !         Wav(*) is the elipse
2230  ! Process:Generate the tolerance elipse @ min max waveforms
2240  !
2250  ! Y=-k +/-V * SRQT(1-(X-j)^2/(H^2))) Elipse Equation at origin (0,0)/(j,k)
2260  !                                  !  _______2H______
2270  !                                    /               \   ^
2280  ! An elipse in terms of Horiz (H)!    (      (j,k)     )  2V
2290  !        Vert (V)                  ! \_______________/   v
2300  !
2310  REAL High,Low,Realwav,Realgold
2320  INTEGER Elipse_pt,Col_cntr,Gold_ptr,H,V,Length_div
2330  !
2340  ! (H) will be equal to the number of columns needed to span one elipse
2350  ! with one point value calculated for each column.
2360  INPUT "Enter horizontal tolerance in Div",Div_h
2370  Length_div=Length/10! The Golden Waveform has "Length" points therefore
2380                      ! divide by 10 to get points per division.
2390  H=Div_h*Length_div
2400  !
2410  ! (V) will be scaled for a full scale deflection of 0..32640
2420  INPUT "Enter vertical tolerance in Div ",Div_v
2430  V=Div_v*(32640/8)                    ! Vertical full scale is 32640
2440  !                                    ! (32640/8) pts/div SINGLE screen
2450  !
2460  ! Generate the elipse once and save it in an array (use symmetry)
2470  ALLOCATE INTEGER Wav(-H:H)           ! *at (j,k) at origin(0,0)
2480  FOR X=0 TO H                         ! Calculate the elipse
2490    Wav(X)=V*(SQRT(1-(X)^2/(H^2)))   ! and load values into an array
2500    Wav(-X)=Wav(X)
2510  NEXT X
2520  !
2530  CALL Draw_e(Wav(*),H)                ! Show the elipse
2540  !
2550  !************************ Mask generation *********************
2560  ! From a Column (Col cntr) point in the reference waveform, traverse
2570  ! the elipse and record the mins and max values in the limit arrays.
2580  ! This is compute bound with "Length * (elipse dimension)" calculations!!!
2590  !
2600  DISP "Computing the limit waveforms Please wait"
2610  FOR Col_cntr=1 TO Length             ! Move across the golden waveform
2620    FOR Elipse_pt=-H TO H              ! with an elipse of +/- (H) points
2630      Gold_ptr=Col_cntr+Elipse_pt      ! Scan a segment of the golden wf
2640      IF Gold_ptr>0 AND Gold_ptr<=Length THEN  ! Watch out for end points,
2650                                       ! need REALs to prevent INT overflow
2660        Realwav=Wav(Elipse_pt)         ! The + value of the elipse
2670        Realgold=Gold_wf(Col_cntr)   ! The REAL value of the Gold wf @ elipse pt
2680        High=Realgold+Realwav          ! /
2690        Low=Realgold-Realwav           ! \
2700        IF High>32640 THEN High=32640    ! Clipped on top?
```

```
2710         IF Low<0 THEN Low=0              ! clipped on bottom?
2720         !
2730         IF Goldmax(Gold_ptr)<High THEN Goldmax(Gold_ptr)=High
2740         IF Goldmin(Gold_ptr)>Low THEN Goldmin(Gold_ptr)=Low
2750         !
2760         ! This routine is COMPUTE BOUND" so the graphics is included to
2770         ! help show activity. performance can be improved slightly by
2780         ! commenting out the draw routines.
2790         !
2800         VIEWPORT 30,100*RATIO,15,85
2810         WINDOW 1,500,0,32640
2820         GOTO Plot_wf
2830         PEN 3
2840         MOVE Gold_ptr,Goldmin(Gold_ptr)
2850         DRAW Gold_ptr,Goldmin(Gold_ptr)
2860         PEN 4
2870         MOVE Gold_ptr,Goldmax(Gold_ptr)
2880         DRAW Gold_ptr,Goldmax(Gold_ptr)
2890 Plot_wf: !
2900         PEN 5
2910         MOVE Gold_ptr,Gold_wf(Gold_ptr)
2920         DRAW Gold_ptr,Gold_wf(Gold_ptr)
2930       END IF
2940     NEXT Elipse_pt
2950   NEXT Col_cntr
2960   DISP ""
2970   DEALLOCATE Wav(*)
2980 SUBEND
```

D

Quick Reference Guide

This section lists the commands and queries with their corresponding arguments and returned formats. The arguments for each command list the minimum argument required. The part of the command or query listed in uppercase letters refers to the short form of that command or query. The long form is the combination of uppercase and lowercase letters.

The following definitions are used:

<block data> ::= definite length block data
<integer> ::= integer - NR1 format
<real number> ::= exponential - NR3 format
<string> ::= string of alphanumeric characters
<measurement> ::= name of measurement
<upper> ::= high limit value
<lower> ::= low limit value

The suffix multipliers available for arguments are:

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

For more information on specific commands or queries, refer to the specific command or query in the programmer's guide.

Command	Argument	Returned Format
Common Commands		
*CLS	—	—
ESE	{0—255}	—
ESE?	—	{0—255}
ESR?	—	{0—255}
IDN?	—	<string>
LRN?	—	<block data>
OPC	—	—
OPC?	—	{1}
OPT?	—	<string>
RCL	{0—4}	—
RST	—	—
SAV	{1—4}	—
SRE	{0—255}	—
SRE?	—	{0—255}
STB?	—	{0—255}
TRG	—	—
TST?	—	<integer>
WAI	—	—

Command	Argument	Returned Format
Root Level Commands		
:AUToscale	—	—
:BEEPER	{ON or 1} or {OFF or 0}	—
:BEEPer?	—	{1 or 0}
:BLANk	{CHANnel 1—4}	—
	{FUNCtion 1 or 2}	—
	{WMEMory 1—4}	—
	{PMEMory 0}	—
:BNC	PROBe	—
	TRIGger	—
:BNC?	—	{PROBe}
	—	{TRIGger}
:DIGitize	{CHANnel 1—4}	—
:ERASe	{PMEMory 0}	—
:LTER?	—	{1 or 0}
:RUN	—	—
:RUN?	—	{1 or 0}
:SERial	<string>	—
:STATus?	{CHANnel 1—4}	{1 or 0}
	{FUNCtion 1 or 2}	{1 or 0}
	{WMEMory 1—4}	{1 or 0}
	{PMEMory 0}	{1 or 0}
:STOP	—	—
:STORe	{CHANnel 1 - 4},{WMEMory 1—4}	—
	{FUNCtion 1 or 2}, {WMEMory 1—4}	—
	{WMEMory 1—4}, {WMEMory 1—4}	—
:TER?	—	{1 or 0}
:VIEW	{CHANnel 1—4}	—
	{FUNCtion 1 or 2}	—
	{WMEMory 1—4}	—
	{PMEMory 0}	—

Command	Argument	Returned Format
System Subsystem Commands		
:SYSTem:COMMunicate:GPIB[:STATe]	{ON or 1} or {OFF or 0}	—
:SYSTem:COMMunicate:GPIB[STATe]?	—	{1 or 0}
:SYSTem:ERRor?	—	<integer>
	{NUMBer}	<integer>
	{STRing}	<integer>,<string>
:SYSTem:HEADer	{OFF or 0}	—
:SYSTem:HEADer?	—	{0}
:SYSTem:LONGform	{ON or 1}	—
	{OFF or 0}	—
:SYSTem:LONGform?	—	{1 or 0}
:SYSTem:SETup	<block data>	—
:SYSTem:SETup?	—	<block data>
Acquire Subsystem Commands		
:ACQuire:COMPlete	{0—100}	—
:ACQuire:COMPlete?	—	{0—100}
:ACQuire:COUNt	{1—2048}	—
:ACQuire:COUNt?	—	{1—2048}
:ACQuire:POINts	{32—1024}	—
:ACQuire:POINts?	—	{32—1024}
:ACQuire:TYPE	{NORMal}	—
	{AVERage}	—
	{ENVelope}	—
:ACQuire:TYPE?	—	{NORMal}
	—	{AVERage}
	—	{ENVelope}

Command	Argument	Returned Format
Calibrate Subsystem Commands		
:CALibrate:PCALibration :ATTenuation:BCALibration	—	—
:CALibrate:PCALibration :ATTenuation:CHANnel{1—4}	—	—
:CALibrate:PCALibration :TNULl:CH1TO{2—4}	{—50nS to 70nS}	—
:CALibrate:REPort?	{CHANnel 1—4}	{CHANnel1 A/D {P|F|D|C}, Gain {P|F|D|C},Offset {P|F|D|C}, Hysteresis {P|F|D|C},Trigger {P|F|D|C}, Delay {P|F|D|C}, Logic Trigger {P|F|D|C}|CHANnel {2|3|4} A/D {P|F|D|C}, Gain {P|F|D|C},Offset {P|F|D|C}, Hysteresis {P|F|D|C}, Trigger {P|F|D|C}, Delay {P|F|D|C}, Time Null {P|F|D|C}
:CALibrate:SCALibration :BCALibration	—	—
:CALibrate:SCALibration :DCALibration	—	—
:CALibrate:SCALibration:DELay	{CHANnel 1—4}	—
:CALibrate:SCALibration:DOUTput	{ZVOLt or FVLOt}	—
:CALibrate:SCALibration :LTCalibrate	{— or ALL or PART}	—
:CALibrate:SCALibration:TNULl	{CH1TO{2|3|4}}	—
:CALibrate:SCALibration :VERTical	—	—
:CALibrate:SECurity:STATe	{ON or 1}| {OFF or 0}	—
:CALibrate:SECurity:STATe?	—	{1 or 0}

Command	Argument	Returned Format
:CALibrate:TNULl	<real number>,<real number>, <real number>	—
:CALibrate:TNULl?	—	<real number>,<real number>, <real number>
Channel Subsystem Commands		
:CHANnel{1—4}:COUPling	{AC} {DC} {DCFifty}	— — —
:CHANnel{1—4}:COUPling?	— — —	{AC} {DC} {DCFifty}
:CHANnel{1—4}:ECL	—	—
:CHANnel{1—4}:HFReject	{ON or 1} {OFF or 0}	— —
:CHANnel{1—4}:HFReject?	—	{1 or 0}
:CHANnel{1—4}:LFReject	{ON or 1} {OFF or 0}	— —
:CHANnel{1—4}:LFReject?	—	{1 or 0}
:CHANnel{1—4}:OFFSet	<real number>	—
:CHANnel{1—4}:OFFSet?	—	<real number>
:CHANnel{1—4}:PROBe	{0.9—1000}	—
:CHANnel{1—4}:PROBe?	—	{0.9—1000}
:CHANnel{1—4}:RANGe	<real number>	—
:CHANnel{1—4}:RANGe?	—	<real number>
:CHANnel{1—4}:TTL	—	—

Command	Argument	Returned Format
Display Subsystem Commands		
:DISPlay:CONNect	{ON or 1} or {OFF or 0}	—
:DISPlay:CONNect	—	{1 or 0}
:DISPlay:FORMat	{1 or 2 or 4}	—
:DISPlay:FORMat?	—	{1 or 2 or 4}
:DISPlay:GRATicule	{OFF} {FRAMe} {AXIS} {GRID}	— — — —
:DISPlay:GRATicule?	— — — —	{OFF} {FRAMe} {AXIS} {GRID}
:DISPlay:PERSistence	<real number> {INFinite} {SINGle}	— — —
:DISPlay:PERSistence?	—	<real number>
:DISPlay:SCReen	{ON or 1} or {OFF or 0}	—
:DISPlay:SCReen?	—	{1 or 0}
:DISPlay:SCReen:ADVisory	{ON or 1} or {OFF or 0}	—
:DISPlay:SCReen:ADVisory?	—	{1 or 0}
:DISPlay:SCReen:IDENtifier	{ON or 1} or {OFF or 0}	—
:DISPlay:SCReen:IDENtifier?	—	{1 or 0}
:DISPlay:SCReen:MEASure	{AUTO} {ON}	— —
:DISPlay:SCReen:MEASure?	— —	{AUTO} {ON}
:DISPlay:SCReen:MEASure:LINE	<integer>	—

Command	Argument	Returned Format
:DISPlay:SCReen:MEASure:LINE?	—	<integer>
:DISPlay:SCReen:STATus	{ON or 1} or {OFF or 0}	—
:DISPlay:SCReen:STATus?	—	{1 or 0}
:DISPlay:SCReen:TIMebase	{ON or 1} or {OFF or 0}	—
:DISPlay:SCReen:TIMebase?	—	{1 or 0}
:DISPlay:TMARker	{ON or 1} or {OFF or 0}	—
:DISPlay:VMARker?	—	{1 or 0}
:DISPlay:VMARker	{ON or 1} or {OFF or 0}	—
:DISPlay:TMARker?	—	{1 or 0}

Command	Argument	Returned Format
Function Subsystem Commands		
:FUNCtion{1 or 2}:ADD	{CHANnel 1—4},{CHANnel 1—4}	—
	{CHANnel 1—4},{WMEMory 1—4}	—
	{WMEMory 1—4},{CHANnel 1—4}	—
	{WMEMory 1—4}, {WMEMory 1—4}	—
:FUNCtion{1 or 2}:INVert	{CHANnel 1—4}	—
	{WMEMory 1—4}	—
:FUNCtion{1 or 2}:MULTiply	{CHANnel 1—4},{CHANnel 1—4}	—
	{CHANnel 1—4},{WMEMory 1—4}	—
	{WMEMory 1—4},{CHANnel 1—4}	—
	{WMEMory 1—4}, {WMEMory 1—4}	—
:FUNCtion{1 or 2}:OFFSet	<real number>	—
:FUNCtion{1 or 2}:OFFSet?	—	<real number>
:FUNCtion{1 or 2}:ONLY	{CHANnel 1—4}	—
	{WMEMory 1—4}	—
:FUNCtion{1 or 2}:RANGe	<real number>	—
:FUNCtion{1 or 2}:RANGe?	—	<real number>
:FUNCtion{1 or 2}:SUBTract	{CHANnel 1—4},{CHANnel 1—4}	—
	{CHANnel 1—4},{WMEMory 1—4}	—
	{WMEMory 1—4},{CHANnel 1—4}	—
	{WMEMory 1—4}, {WMEMory 1—4}	—
:FUNCtion{1 or 2}:VERSus	{CHANnel 1—4},{CHANnel 1—4}	—
	{CHANnel 1—4},{WMEMory 1—4}	—
	{WMEMory 1—4},{CHANnel 1—4}	—
	{WMEMory 1—4}, {WMEMory 1—4}	—

Command	Argument	Returned Format
Measure Subsystem Commands		
:MEASure:ALL?	—	<real number>, . . .
:MEASure:COMPare	<measurement>,<upper>, <lower>	—
:MEASure:COMPare?	<measurement>	<real number>,<real number>
:MEASure:CURSor?	{DELTa}	<real number>, <real number>
	{STARt}	<real number>, <real number>
	{STOP}	<real number>, <real number>
:MEASure:DEFine	{DELay},<real number>, . . .	—
	{PWIDth},{MIDDle}	—
	{PWIDth},{UPPer}	—
	{PWIDth},{LOWer}	—
	{NWIDth},{MIDDle}	—
	{NWIDth},{UPPer}	—
	{NWIDth},{LOWer}	—
:MEASure:DEFine?	{DELay}	<real number>, . . .
	{PWIDth}	{MIDDle}
	{PWIDth}	{UPPer}
	{PWIDth}	{LOWer}
	{NWIDth}	{MIDDle}
	{NWIDth}	{UPPer}
	{NWIDth}	{LOWer}
:MEASure:DELay	—	—
:MEASure:DELay?	—	<real number>
:MEASure:DESTination	{WMEMory 1—4}	—
	{OFF}	—
:MEASure:DESTination?	—	{WMEMory 1—4}
	—	{OFF}
:MEASure:DUTycycle	—	—
:MEASure:DUTycycle?	—	<real number>
:MEASure:ESTArt	<+ or –> <integer>	—
:MEASure:ESTArt?	—	<integer>
:MEASure:ESTOp	<+ or –> <integer>	—

Command	Argument	Returned Format
:MEASure:ESTOp?	—	<integer>
:MEASure:FALLtime	—	—
:MEASure:FALLtime?	—	<real number>
:MEASure:FREQuency	—	—
:MEASure:FREQuency?	—	<real number>
:MEASure:LIMittest	{MEASure} {OFF}	— —
:MEASure:LOWer	<real number>	—
:MEASure:LOWer?	—	<real number>
:MEASure:MODE	{STANdard} {USER}	— —
:MEASure:MODE?	— —	{STANdard} {USER}
:MEASure:NWIDth	—	—
:MEASure:NWIDth?	—	<real number>
:MEASure:OVERshoot	—	—
:MEASure:OVERshoot?	—	<real number>
:MEASure:PERiod	—	—
:MEASure:PERiod?	—	<real number>
:MEASure:POSTfailure	{CONTinue} {STOP}	— —
:MEASure:POSTfailure?	— —	{CONTinue} {STOP}
:MEASure:PRECision	{COARse}	—
:MEASure:PRECision?	—	{COARse}

Command	Argument	Returned Format
:MEASure:PREShoot	—	—
:MEASure:PREShoot?	—	<real number>
:MEASure:PWIDth	—	—
:MEASure:PWIDth?	—	<real number>
:MEASure:RESults?	— —	{1—8}<string>[; <string>] . . . {0}
:MEASure:RISetime	—	—
:MEASure:RISetime?	—	<real number>
:MEASure:SCRatch	—	—

:MEASure:SOURce	{CHANnel 1—4}
	{FUNCtion 1 or 2}
	{WMEMory 1—4}
		—
		—
		—

:MEASure:SOURce?	—
	—
	—
		{CHANnel 1—4}
		{FUNCtion 1 or 2}
		{WMEMory 1—4}

:MEASure:STATistics	{ON or 1} {OFF or 0}	— —
:MEASure:STATistics?	—	{1 or 0}
:MEASure:TDELta?	—	<real number>
:MEASure:TMAX?	—	<real number>
:MEASure:TMIN?	—	<real number>
:MEASure:TSTArt	<real number>	—
:MEASure:TSTArt?	—	<real number>
:MEASure:TSTOp	<real number>	—
:MEASure:TSTOp?	—	<real number>

Command	Argument	Returned Format
:MEASure:TVOLt?	<real number>,{+ or −} <integer>	<real number>
:MEASure:UNITs	{PERCent}	—
	{VOLTs}	—
:MEASure:UNITs?	—	{PERCent}
	—	{VOLTs}
:MEASure:UPPer	<real number>	—
:MEASure:UPPer?	—	<real number>
:MEASure:VACRms	—	—
:MEASure:VACRms?	—	<real number>
:MEASure:VAMPlitude	—	—
:MEASure:VAMPlitude?	—	<real number>
:MEASure:VAVerage	—	—
:MEASure:VAVerage?	—	<real number>
:MEASure:VBASe	—	—
:MEASure:VBASe?	—	<real number>
:MEASure:VDCRms	—	—
:MEASure:VDCRms?	—	<real number>
:MEASure:VDELta?	—	<real number>
:MEASure:VFIFty	—	—
:MEASure:VMAX	—	—
:MEASure:VMAX?	—	<real number>
:MEASure:VMIN	—	—
:MEASure:VMIN?	—	<real number>

Command	Argument	Returned Format
:MEASure:VPP	—	—
:MEASure:VPP?	—	<real number>
:MEASure:VRELative	{0—100}	—
:MEASure:VRELative?	—	{50—100}
:MEASure:VRMS	—	—
:MEASure:VRMS?	—	<real number>
:MEASure:VSTArt	<real number>	—
:MEASure:VSTArt?	—	<real number>
:MEASure:VSTOp	<real number>	—
:MEASure:VSTOp?	—	<real number>
:MEASure:VTIMe?	<real number>	<real number>
:MEASure:VTOP	—	—
:MEASure:VTOP?	—	<real number>

Command	Argument	Returned Format
Summary Subsystem Commands		
:SUMMary:PRESet	—	—
:SUMMary:QUESTionable:CONDition?	—	<register contents>
:SUMMary:QUESTionable: ENABle	<number>	—
:SUMMary:QUESTionable:ENABle?	—	<register contents>
:SUMMary:QUESTionable:EVENt?	—	<register contents>
:SUMMary:QUESTionable :CALibration:CONDition?	—	<register contents>
:SUMMary:QUESTionable :CALibration:ENABle	<number>	—
:SUMMary:QUESTionable :CALibration:ENABle?	—	<register contents>
:SUMMary:QUESTionable :CALibration:EVENt?	—	<register contents>
:SUMMary:QUESTionable :CALibration:CHANnel {1|2|3|4}:CONDition?	—	<register contents>
:SUMMary:QUESTionable :CALibration:CHANnel {1|2|3|4}:ENABle	<number>	—
:SUMMary:QUESTionable :CALibration:CHANnel {1|2|3|4}:ENABle?	—	<register contents>
:SUMMary:QUESTionable :CALibration:CHANnel {1|2|3|4}:EVENt?	—	<register contents>
:SUMMary:QUESTionable :CALibration:CHANnel {1|2|3|4}:AD:CONDition?	—	<register contents>
:SUMMary:QUESTionable :CALibration:CHANnel {1|2|3|4}:AD:ENABle	<number>	—
:SUMMary:QUESTionable :CALibration:CHANnel {1|2|3|4}:AD:ENABle?	—	<register contents>

Command	Argument	Returned Format
:SUMMary:QUESTionable :CALibration:CHANnel {1|2|3|4}:EVENt?	—	<register contents>
:SUMMary:QUESTionable :CALibration:CHANnel {1|2|3|4}:DELay:CONDition?	—	<register contents>
:SUMMary:QUESTionable :CALibration:CHANnel {1|2|3|4}:DELay:ENABle	<number>	—
:SUMMary:QUESTionable :CALibration:CHANnel {1|2|3|4}:DELay:ENABle?	—	<register contents>
:SUMMary:QUESTionable :CALibration:CHANnel {1|2|3|4}:DELay:EVENt?	—	<register contents>
:SUMMary:QUESTionable :CALibration:CHANnel {1|2|3|4}:GAIN:CONDition?	—	<register contents>
:SUMMary:QUESTionable :CALibration:CHANnel {1|2|3|4}:GAIN:ENABle	<number>	—
:SUMMary:QUESTionable :CALibration:CHANnel {1|2|3|4}:GAIN:ENABle?	—	<register contents>
:SUMMary:QUESTionable :CALibration:CHANnel {1|2|3|4}:GAIN:EVENt?	—	<register contents>
:SUMMary:QUESTionable :CALibration:CHANnel{1|2|3|4} :HYSTeresis:CONDtion?	—	<register contents>
:SUMMary:QUESTionable :CHANnel{1|2|3|4}:HYSTeresis :ENABle	<number>	—
:SUMMary:QUESTionable :CALibration:CHANnel{1|2|3|4} :HYSTeresis:ENABle?	—	<register contents>
:SUMMary:QUESTionable :CALibration:CHANnel{1|2|3|4} :HYSTeresis:EVENt	—	<register contents>

Command	Argument	Returned Format
:SUMMary:QUESTionable :CALibration:CHANnel1 :LTRigger:CONDition?	—	<register contents>
:SUMMary:QUESTionable :CALibration:CHANnel1 :LTRigger:ENABle	<number>	—
:SUMMary:QUESTionable :CALibration:CHANnel1 :LTRigger:ENABle?	—	<register contents>
:SUMMary:QUESTionable :CALibration:CHANnel1 :LTRigger:EVENt?	—	<register contents>
:SUMMary:QUESTionable :CALibration:CHANnel{1|2|3|4} :OFFSet:CONDtion?	—	<register contents>
:SUMMary:QUESTionable :CALibration:CHANnel{1|2|3|4} :OFFSet:ENABle	<number>	—
:SUMMary:QUESTionable :CALibration:CHANnel{1|2|3|4} :OFFSet:ENABle?	—	—
:SUMMary:QUESTionable :CALibration:CHANnel{1|2|3|4} :OFFSet:EVENt?	—	—
:SUMMary:QUESTionable :CALibration:CHANnel {2|3|4}:TNULl :CONDition?	—	<register contents>
:SUMMary:QUESTionable :CALibration:CHANnel {2|3|4}:TNULl: ENABle	<number>	—
:SUMMary:QUESTionable :CALibration:CHANnel {2|3|4}:TNULl :ENABle?	—	<register contents>
:SUMMary:QUESTionable :CALibration:CHANnel {2|3|4}:TNULl:EVENt?	—	<register contents>

Command	Argument	Returned Format
:SUMMary:QUESTionable :CALibration:CHANnel<N> :TRIGger:CONDition?	—	<register contents>
:SUMMary:QUESTionable :CALibration:CHANnel<N> :TRIGger:ENABle	<number>	—
:SUMMary:QUESTionable :CALibration:CHANnel<N> :TRIGger:ENABle?	—	<register contents>
:SUMMary:QUESTionable :CALibration:CHANnel<N> :TRIGger:EVENt?	—	<register contents>
:SUMMary:QUESTionable :CALibration:DCALibration :CONDition?	—	<register contents>
:SUMMary:QUESTionable :CALibration:DCALibration :ENABle	<number>	—
:SUMMary:QUESTionable :CALibration:DCALibration :ENABle?	—	<register contents>
:SUMMary:QUESTionable :CALibration:DCALibration :EVENt?	—	<register contents>
:SUMMary:QUESTionable :CALibration:PROBe :CONDition?	—	<register contents>
:SUMMary:QUESTionable :CALibration:PROBe:ENABle	<number>	—
:SUMMary:QUESTionable :CALibration:PROBe:ENABle?	—	<register contents>
:SUMMary:QUESTionable :CALibration:PROBe:EVENt	—	<register contents>
:SUMMary:QUESTionable :TEST:CONDition?	—	<register contents>

Command	Argument	Returned Format
:SUMMary:QUESTionable :TEST:ENABle	<number>	—
:SUMMary:QUESTionable :TEST:ENABle?	—	<register contents>
:SUMMary:QUESTionable :TEST:EVENt?	—	<register contents>
:SUMMary:QUESTionable:TEST :ACQuisition:CONDition	—	<register contents>
:SUMMary:QUESTionable:TEST :ACQuisition:ENABle	<number>	—
:SUMMary:QUESTionable:TEST :ACQuisition:ENABle?	—	<register contents>
:SUMMary:QUESTionable:TEST :ACQuisition:EVENt?	—	<register contents>
:SUMMary:QUESTionable:TEST :ACQuisition:AD:CONDition?	—	<register contents>
:SUMMary:QUESTionable:TEST :ACQuisition:AD:ENABle	<number>	—
:SUMMary:QUESTionable:TEST :ACQuisition:AD:ENABle?	—	<register contents>
:SUMMary:QUESTionable:TEST :ACQuisition:AD:EVENt?	—	<register contents>
:SUMMary:QUESTionable:TEST :ACQuisition:ATRigger:CONDition?	—	<register contents>
:SUMMary:QUESTionable:TEST :ACQuisition:ATRigger:ENABle	<number>	—
:SUMMary:QUESTionable:TEST :ACQuisition:ATRigger:ENABle?	—	<register contents>
:SUMMary:QUESTionable:TEST :ACQuisition:ATRigger:EVENt?	—	<register contents>
:SUMMary:QUESTionable:TEST :ACQuisition:DA:CONDtion?	—	<register contents>
:SUMMary:QUESTionable:TEST :ACQuisition:DA:ENABle	<number>	—

Command	Argument	Returned Format
:SUMMary:QUESTionable:TEST :ACQuisition:DA:ENABle?	—	<register contents>>
:SUMMary:QUESTionable:TEST :ACQuisition:DA:EVENt?	—	<register contents>
:SUMMary:QUESTionable:TEST :ACQuisition:LTRigger:CONDition?	—	<register contents>
:SUMMary:QUESTionable:TEST :ACQuisition:LTRigger:ENABle	<number>	—
:SUMMary:QUESTionable:TEST :ACQuisition:LTRigger:ENABle?	—	<register contents>
:SUMMary:QUESTionable:TEST :ACQuisition:LTRigger:EVENt?	—	<register contents>
:SUMMary:QUESTionable:TEST :ACQuisition:TIMebase:CONDition?	—	<register contents>
:SUMMary:QUESTionable:TEST :ACQuisition:TIMebase:ENABle	<number>	—
:SUMMary:QUESTionable:TEST :ACQuisition:TIMebase:ENABle?	—	<register contents>
:SUMMary:QUESTionable:TEST :ACQuisition:TIMebase:EVENt?	—	<register contents>
:SUMMary:QUESTionable:TEST :ACQuisition:TIMebase:INTerpolator :CONDition?	—	<register contents>
:SUMMary:QUESTionable:TEST :ACQuisition: TIMebase:INTerpolator :ENABle	<number>	—
:SUMMary:QUESTionable:TEST :ACQuisition:TIMebase:INTerpolator :ENABle?	—	<register contents>
:SUMMary:QUESTionable:TEST :ACQuisition:TIMebase:INTerpolator :EVENt?	—	<register contents>

Command	Argument	Returned Format
:SUMMary:QUESTionable:TEST :RAM:CONDition?	—	<register contents>
:SUMMary:QUESTionable:TEST :RAM:ENABle	<number>	—
:SUMMary:QUESTionable:TEST :RAM:ENABle?	—	<register contents>
:SUMMary:QUESTionable:TEST :RAM:EVENt?	—	<register contents>
:SUMMary:QUESTionable:TEST :RAM:ACQuisition:CONDition?	—	<register contents>
:SUMMary:QUESTionable:TEST :RAM:ACQuisition: ENABle	<number>	—
:SUMMary:QUESTionable:TEST :RAM:ACQuisition: ENABle?	—	<register contents>
:SUMMary:QUESTionable:TEST :RAM:ACQuisition: EVENt?	—	<register contents>
:SUMMary:QUESTionable:TEST :RAM:DISPlay:CONDtion?	—	<register contents>
:SUMMary:QUESTionable:TEST :RAM:DISPlay:ENABle	<number>	—
:SUMMary:QUESTionable:TEST :RAM:DISPlay:ENABle?	—	<register contents>
:SUMMary:QUESTionable:TEST :RAM:DISPlay:EVENt?	—	<register contents>
SUMMary:QUESTionable:TEST :RAM:NVOLatile:CONDition?	—	<register contents>
:SUMMary:QUESTionable:TEST :RAM:NVOLatile:ENABle	<number>	—
:SUMMary:QUESTionable:TEST :RAM:NVOLatile:ENABle?	—	<register contents>
:SUMMary:QUESTionable:TEST :RAM:NVOLatile:EVENt?	—	<register contents>

Command	Argument	Returned Format
:SUMMary:QUESTionable:TEST :RAM:SYSTem:CONDtion?	—	<register contents>
:SUMMary:QUESTionable:TEST :RAM:SYSTem:ENABle	<number>	—
:SUMMary:QUESTionable:TEST :RAM:SYSTem:ENABle?	—	<register contents>
:SUMMary:QUESTionable:TEST :RAM:SYSTem:EVENt?	—	<register contents>
:SUMMary:QUESTionable :TEST:ROM:CONDition?	—	<register contents>
:SUMMary:QUESTionable :TEST:ROM:ENABle	<number>	—
:SUMMary:QUESTionable :TEST:ROM:ENABle?	—	<register contents>
:SUMMary:QUESTionable :TEST:ROM:EVENt?	—	<register contents>
:SUMMary:QUESTionable:TEST:ROM :NPRotect:CONDition?	—	<register contents>
:SUMMary:QUESTionable:TEST:ROM :NPRotect:ENABle	<number>	—
:SUMMary:QUESTionable:TEST:ROM :NPRotect:ENABle?	—	<register contents>
:SUMMary:QUESTionable:TEST:ROM :NPRotect:EVENt	—	<register contents>
:SUMMary:QUESTionable:TEST:ROM :SYSTem:CONDition?	—	<register contents>
:SUMMary:QUESTionable:TEST:ROM :SYSTem:ENABle	<number>	—
:SUMMary:QUESTionable:TEST:ROM :SYSTem:ENABle?	—	<register contents>
:SUMMary:QUESTionable :TEST:ROM:SYSTem:EVENt?	—	<register contents>

Command	Argument	Returned Format
:SUMMary:QUESTionable :TIME:CONDition?	—	<register contents>
:SUMMary:QUESTionable :TIME:ENABle	<number>	—
:SUMMary:QUESTionable :TIME:ENABle?	—	<register contents>
:SUMMary:QUESTionable :TIME:EVENt?	—	<register contents>
Test Subsystem Commands		
:TEST:ACQ	{ATR | LTR | AD | TIM | DA}	—
:TEST:RAM	{DISP | ACQ | SYST | NVOL}	—
:TEST:ROM	{SYST | NVP}	—
:TEST:TALL	—	—

Command	Argument	Returned Format
Timebase Subsystem Commands		
:TIMebase:DELay	<real number>	—
:TIMebase:DELay?	—	<real number>
:TIMebase:MODE	{AUTO}	—
	{TRIGgered}	—
	{SINGle}	—
:TIMebase:MODE?	—	{AUTO}
	—	{TRIGgered}
	—	{SINGle}
:TIMebase:RANGe	<real number>	—
:TIMebase:RANGe?	—	<real number>
:TIMebase:REFerence	{LEFT}	—
	{CENTer}	—
	{RIGHt}	—
:TIMebase:REFerence?	—	{LEFT}
	—	{CENTer}
	—	{RIGHt}
:TIMebase:WINDow	{ON or 1}	—
	{OFF or 0}	—
:TIMebase:WINDow?	—	{1 or 0}
:TIMebase:WINDow:DELay	<real number>	—
:TIMebase:WINDow:DELay?	—	<real number>
:TIMebase:WINDow:RANGe	<real number>	—
:TIMebase:WINDow:RANGe?	—	<real number>

Command	Argument	Returned Format
Trigger Subsystem Commands		
:TRIGger:CENTered	—	—
:TRIGger:CONDition	{ENTer}	—
	{EXIT}	—
	{GT},<real number>	—
	{LT},<real number>	—
	{RANGe},<real number>, <real number>	—
	{TRUE}	—
	{FALSe}	—
TRIGger:CONDition?	—	{ENTer}
	—	{EXIT}
	—	{GT},<real number>
	—	{LT},<real number>
	—	{RANGe},<real number>, <real number>
	—	{TRUE}
	—	{FALSe}
:TRIGger:DELay	{TIME},<real number>	—
	{EVENt},<real number>	—
:TRIGger:DELay?	—	{TIME},<real number>
	—	{EVENt},<real number>
:TRIGger:DELay:SLOPe	{POSitive}	—
	{NEGative}	—
:TRIGger:DELay:SLOPe?	—	{POSitive}
	—	{NEGative}
:TRIGger:DELay:SOURce	{CHANnel 1—4}	—
:TRIGger:DELay:SOURce?	—	{CHANnel 1—4}
:TRIGger:FIELd	{1 or 2}	—
:TRIGger:FIELd?	—	{1 or 2}
:TRIGger:HOLDoff	{TIME},<real number>	—
	{EVENt},{1—16000000}	—

Command	Argument	Returned Format
:TRIGger:HOLDoff?	—	{TIME},<real number>
	—	{EVENt},<integer>
:TRIGger:LEVel	<real number>	—
:TRIGger:LEVel?	—	<real number>
:TRIGger:LINE	{1—625}	—
:TRIGger:LINE?	—	{1—625}
:TRIGger:LOGic	{HIGH}	—
	{LOW}	—
	{DONTcare}	—
:TRIGger:LOGic?	—	{HIGH}
	—	{LOW}
	—	{DONTcare}
:TRIGger:MODE	{EDGE}	—
	{PATTern}	—
	{STATe}	—
	{DELay}	—
	{TV}	—
:TRIGger:MODE?	—	{EDGE}
	—	{PATTern}
	—	{STATe}
	—	{DELay}
	—	{TV}
:TRIGger:OCCurrence	{1—16000000}	—
:TRIGger:OCCurrence?	—	{1—16000000}
:TRIGger:OCCurrence:SLOPe	{POSitive}	—
	{NEGative}	—
:TRIGger:OCCurrence:SLOPe?	—	{POSitive}
	—	{NEGative}
:TRIGger:OCCurrence:SOURce	{CHANnel 1—4}	—
:TRIGger:OCCurrence:SOURce?	—	{CHANnel 1—4}

Command	Argument	Returned Format
:TRIGger:PATH	{CHANnel 1—4}	—
:TRIGger:PATH?	—	{CHANnel 1—4}
:TRIGger:POLarity	{POSitive}	—
	{NEGative}	—
:TRIGger:POLarity?	—	{POSitive}
	—	{NEGative}
:TRIGger:QUALity	{EDGE}	—
	{PATTern}	—
	{STATe}	—
	{LOW}	—
	{HIGH}	—
:TRIGger:QUALity?	—	{EDGE}
	—	{PATTern}
	—	{STATe}
	—	{LOW}
	—	{HIGH}
:TRIGger:SENSitivity	{NORMal}	—
	{LOW}	—
:TRIGger:SENSitivity?	—	{NORMal}
	—	{LOW}
:TRIGger:SLOPe	{POSitive}	—
	{NEGative}	—
:TRIGger:SLOPe?	—	{POSitive}
	—	{NEGative}
:TRIGger:SOURce	{CHANnel 1—4}	—
:TRIGger:SOURce?	—	{CHANnel 1—4}
:TRIGger:STANdard	{525}	—
	{625}	—
	{USER}	—
:TRIGger:STANdard?	—	{525}
	—	{625}
	—	{USER}

Command	Argument	Returned Format
Waveform Subsystem Commandsj		
:WAVeform:COUNt?	—	{1}
:WAVeform:DATA	<block data>	—
:WAVeform:DATA?	—	<block data>
:WAVeform:FORMat	{WORD}	—
	{BYTE}	—
	{COMPressed}	—
:WAVeform:FORMat?	—	{WORD}
	—	{BYTE}
	—	{COMPressed}
:WAVeform:POINts?	—	<integer>
:WAVeform:PREamble	<real number>, . . .	—
:WAVeform:PREamble?	—	<real number>, . . .
:WAVeform:SOURce	{CHANnel 1—4}	—
	{WMEMory 1—4}	—
:WAVeform:SOURce?	—	{CHANnel 1—4}
	—	{WMEMory 1—4}
:WAVeform:TYPE?	—	{AVERage}
	—	{ENVelope}
	—	{NORMal}
:WAVeform:XINCrement?	—	<real number>
:WAVeform:XORigin?	—	<real number>
:WAVeform:XREFerence?	—	{0}
:WAVeform:YINCrement?	—	<real number>
:WAVeform:YORigin?	—	<real number>
:WAVeform:YREFerence?	—	<integer>

Index