Programmer's Guide

infiniium DCA

Agilent 86100A

Agilent Technologies

Agilent Part No. 86100-90017
Printed in USA
November 2000

Agilent Technologies
Lightwave Division
1400 Fountaingrove Parkway
Santa Rosa, CA 95403-1799, USA

Notice.
The information contained in this document is subject to change without notice. Companies, names, and data used in examples herein are fictitious unless otherwise noted. Agilent Technologies makes no warranty of any kind with regard to this material, including but not limited to, the implied warranties of merchantability and fitness for a particular purpose. Agilent Technologies shall not be liable for errors contained herein or for incidental or consequential damages in connection with the furnishing, performance, or use of this material.

Restricted Rights Legend.
Use, duplication, or disclosure by the U.S. Government is subject to restrictions as set forth in subparagraph (c) (1) (ii) of the Rights in Technical Data and Computer Software clause at DFARS 252.227-7013 for DOD agencies, and subparagraphs (c) (1) and (c) (2) of the Commercial Computer Software Restricted Rights clause at FAR 52.227-19 for other agencies.

Warranty.
This Agilent Technologies instrument product is warranted against defects in material and workmanship for a period of one year from date of shipment. During the warranty period, Agilent Technologies will, at its option, either repair or replace products which prove to be defective. For warranty service or repair, this product must be returned to a service facility designated by Agilent Technologies. Buyer shall prepay shipping charges to Agilent Technologies and Agilent Technologies shall pay shipping charges to return the product to Buyer. However, Buyer shall pay all shipping charges, duties, and taxes for products returned to Agilent Technologies from another country.

Agilent Technologies warrants that its software and firmware designated by Agilent Technologies for use with an instrument will execute its programming instructions when properly installed on that instrument. Agilent Technologies does not warrant that the operation of the instrument, or software, or firmware will be uninterrupted or error-free.

Limitation of Warranty.
The foregoing warranty shall not apply to defects resulting from improper or inadequate maintenance by Buyer, Buyer-supplied software or interfacing, unauthorized modification or misuse, operation outside of the environmental specifications for the product, or improper site preparation or maintenance.

No other warranty is expressed or implied. Agilent Technologies specifically disclaims the implied warranties of merchantability and fitness for a particular purpose.

Exclusive Remedies.
The remedies provided herein are buyer's sole and exclusive remedies. Agilent Technologies shall not be liable for any direct, indirect, special, incidental, or consequential damages, whether based on contract, tort, or any other legal theory.

Safety Symbols.

CAUTION
The *caution* sign denotes a hazard. It calls attention to a procedure which, if not correctly performed or adhered to, could result in damage to or destruction of the product. Do not proceed beyond a caution sign until the indicated conditions are fully understood and met.

WARNING
The *warning* sign denotes a hazard. It calls attention to a procedure which, if not correctly performed or adhered to, could result in injury or loss of life. Do not proceed beyond a warning sign until the indicated conditions are fully understood and met.

⚠ The instruction manual symbol. The product is marked with this warning symbol when it is necessary for the user to refer to the instructions in the manual.

The laser radiation symbol. This warning symbol is marked on products which have a laser output.

∿ The AC symbol is used to indicate the required nature of the line module input power.

| The ON symbols are used to mark the positions of the instrument power line switch.

❍ The OFF symbols are used to mark the positions of the instrument power line switch.

CE The CE mark is a registered trademark of the European Community.

The CSA mark is a registered trademark of the Canadian Standards Association.

ISM1-A This text denotes that the instrument is an Industrial Scientific and Medical Group 1 Class A product.

Typographical Conventions.
The following conventions are used in this book:

key type for keys or text located on the keyboard or instrument.

softkey type for key names that are displayed on the instrument's screen.

`display` type for words or characters displayed on the computer's screen or instrument's display.

user type for words or characters that you type or enter.

emphasis type for words or characters that emphasize some point or that are used as place holders for text that you type.

General Safety Considerations

WARNING **If this product is not used as specified, the protection provided by the equipment could be impaired. This product must be used in a normal condition (in which all means for protection are intact) only.**

WARNING **No operator serviceable parts inside. Refer servicing to qualified personnel. To prevent electrical shock, do not remove covers.**

CAUTION Fiber-optic connectors are easily damaged when connected to dirty or damaged cables and accessories. The digital communications analyzer's front-panel INPUT connector is no exception. When you use improper cleaning and handling techniques, you risk expensive instrument repairs, damaged cables, and compromised measurements. Before you connect any fiber-optic cable to the digital communications analyzer, refer to "Cleaning Connectors for Accurate Measurements" in the *Agilent 86100A Quick Start Guide*, or in the instrument on-line help system.

This product has been designed and tested in accordance with IEC Publication 61010-1, Safety Requirements for Electrical Equipment for Measurement, Control, and Laboratory Use, and has been supplied in a safe condition. The instruction documentation contains information and warnings that must be followed by the user to ensure safe operation and to maintain the product in a safe condition.

Contents

1

Introduction

Introduction to Programming

This chapter introduces the basics for remote programming of an analyzer. The programming commands in this manual conform to the IEEE 488.2 Standard Digital Interface for Programmable Instrumentation. The programming commands provide the means of remote control.

Basic operations that you can do with a computer (GPIB controller) and an analyzer include:

- Set up the analyzer.
- Make measurements.
- Get data (waveform, measurements, configuration) from the analyzer.
- Send information, such as waveforms and configurations, to the analyzer.

Other tasks are accomplished by combining these functions.

Example Programs are Written in HP BASIC and C

The programming examples for individual commands in this manual are written in HP BASIC and C.

Communicating with the Analyzer

Computers communicate with the analyzer by sending and receiving messages over a remote interface, usually with GPIB programming. Commands for programming normally appear as ASCII character strings embedded in the output statements of a “host” language available on your computer. The input commands of the host language are used to read in responses from the analyzer.

For example, HP BASIC uses the OUTPUT statement for sending commands and queries. After a query is sent, the response is usually read using the HP BASIC ENTER statement. The ENTER statement passes the value across the bus to the computer and places it in the designated variable.

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

This HP BASIC OUTPUT statement sends a command that sets the channel 1 scale value to 500 mV:

```
OUTPUT <device address>;":CHANNEL1:SCALE 500E-3"<terminator>
```

The device address represents the address of the device being programmed. Each of the other parts of the above statement are explained in the following pages.

Use the Suffix Multiplier Instead

Using "mV" or "V" following the numeric voltage value in some commands will cause Error 138–Suffix not allowed. Instead, use the convention for the suffix multiplier as described in Chapter 3, “Message Communication and System Functions”.

Output Command

The output command depends entirely on the programming language. Throughout this book, HP BASIC and ANSI C are used in the examples of individual commands. If you are using other languages, you will need to find the equivalents of HP BASIC commands like OUTPUT, ENTER, and CLEAR, to convert the examples.

Device Address

The location where the device address must be specified depends on the programming language you are using. In some languages, it may be specified outside the OUTPUT command. In HP BASIC, it is always specified after the keyword OUTPUT. The examples in this manual assume that the analyzer and interface card are at GPIB device address 707. When writing programs, the device address varies according to how the bus is configured.

Instructions

Instructions, both commands and queries, normally appear as strings embedded in a statement of your host language, such as HP BASIC, Pascal, or C. The only time a parameter is not meant to be expressed as a string is when the instruction's syntax definition specifies <block data>, such as HP BASIC's "learnstring" command. There are only a few instructions that use block data.

Instructions are composed of two main parts:

- The header, which specifies the command or query to be sent.
- The program data, which provides additional information to clarify the meaning of the instruction.

Instruction Header

The instruction header is one or more command mnemonics separated by colons (:) that represent the operation to be performed by the analyzer. See Chapter 5, “Programming Conventions” for more information.

Queries are formed by adding a question mark (?) to the end of the header. Many instructions can be used as either commands or queries, depending on whether or not you include the question mark. The command and query forms of an instruction usually have different program data. Many queries do not use any program data.

White Space (Separator)

White space is used to separate the instruction header from the program data. If the instruction does not require any program data parameters, you do not need to include any white space. In this manual, white space is defined as one or more spaces. ASCII defines a space to be character 32, in decimal.

Program Data

Program data is used to clarify the meaning of the command or query. It provides necessary information, such as whether a function should be on or off or which waveform is to be displayed. Each instruction's syntax definition shows the program data, and the values they accept. See "Numeric Program Data" on page 1-9 for more information about general syntax rules and acceptable values.

When there is more than one data parameter, they are separated by commas (,). You can add spaces around the commas to improve readability.

Header Types

There are three types of headers:

- Simple Command headers
- Compound Command headers
- Common Command headers

Simple Command Header

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

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

or

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

When program data must be included with the simple command header (for example, :DIGITIZE CHAN1), white space is added to separate the data from the header. The syntax is:

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

or

```
OUTPUT 707;":DIGITIZE CHANNEL1,FUNCTION2"
```

Compound Command Header

Compound command headers are a combination of two program mnemonics. The first mnemonic selects the subsystem, and the second mnemonic selects the function within that subsystem. The mnemonics within the compound message are separated by colons. For example:

To execute a single function within a subsystem:

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

For example:

```
OUTPUT 707;":CHANNEL1:BANDWIDTH HIGH"
```

Combining Commands in the Same Subsystem

To execute more than one command within the same subsystem, use a semi-colon (;) to separate the commands:

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

For example:

```
:CHANNEL1:DISPLAY ON;BWLIMIT ON
```

Common Command Header

Common command headers, such as clear status, control the IEEE 488.2 functions within the analyzer. The 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.

Duplicate Mnemonics

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.

To set the vertical range of channel 1 to 0.4 volts full scale:

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

To set the horizontal time base to 1 second full scale:

```
:TIMEBASE:RANGE 1
```

CHANNEL1 and TIMEBASE are subsystem selectors, and determine which range is being modified.

Query Headers

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

```
:TIMEBASE:RANGE?
```

places the current time base setting in the output queue.

In HP BASIC, the computer input statement:

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

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

You can use query commands to find out how the analyzer is currently configured. They are also used to get results of measurements made by the analyzer. For example, the command:

```
:MEASURE:RISETIME?
```

tells the analyzer to measure the rise time of your waveform and place the result in the output queue.

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 it with an input statement. In HP BASIC, this is usually done with an ENTER statement immediately followed by a variable name. This statement reads the result of the query and places the result in a specified variable.

Handling Queries Properly

If you send another command or query before reading the result of a query, the output buffer is cleared and the current response is lost. This also generates a query-interrupted error in the error queue. If you execute an input statement before you send a query, it will cause the computer to wait indefinitely.

Program Header Options

You can send program headers using any combination of uppercase or lowercase ASCII characters. Analyzer responses, however, are always returned in uppercase.

You may send program command and query headers in either long form (complete spelling), short form (abbreviated spelling), or any combination of long form and short form. For example:

:TIMEBASE:DELAY 1E-6 is the long form.

:TIM:DEL 1E-6 is the short form.

Using Long Form or Short Form

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

The rules for the short form syntax are described in Chapter 5, “Programming Conventions”.

Character Program Data

Character program data is used to convey parameter information as alpha or alphanumeric strings. For example, the :TIMEBASE:REFERENCE command can be set to left, center, or right. The character program data in this case may be LEFT, CENTER, or RIGHT. The command :TIMEBASE:REFERENCE RIGHT sets the time base reference to right.

The available mnemonics for character program data are always included with the instruction's syntax definition. Either the long form of commands, or the short form (if one exists), may be sent. Uppercase and lowercase letters may be mixed freely. When receiving responses, uppercase letters are used exclusively.

Numeric Program Data

Some command headers require program data to be expressed numerically. For example, :TIMEBASE:RANGE requires the desired full scale range to be expressed numerically.

For numeric program data, you can use exponential notation or suffix multipliers to indicate the numeric value. The following numbers are all equal:

28 = 0.28E2 = 280E-1 = 28000m = 0.028K = 28E-3K

When a syntax definition specifies that a number is an integer, it means that the number should be whole. Any fractional part is ignored and truncated. Numeric data parameters that accept fractional values are called real numbers. For more information see Chapter 2, “Interface Functions”.

All numbers are expected to be strings of ASCII characters.

- When sending the number 9, you would send a byte representing the ASCII code for the character “9” (which is 57).
- A three-digit number like 102 would take up three bytes (ASCII codes 49, 48, and 50). The number of bytes is figured automatically when you include the entire instruction in a string.

Embedded Strings

Embedded strings contain groups of alphanumeric characters which are treated as a unit of data by the analyzer. An example of this is the line of text written to the advisory line of the analyzer with the :SYSTEM:DSP command:

```
:SYSTEM:DSP ""This is a message.""
```

You may delimit embedded strings with either single (') or double (") quotation marks. These strings are case-sensitive, and spaces act as legal characters just like any other character.

Program Message Terminator

The program instructions within a data message are executed after the program message terminator is received. The terminator may be either a NL (New Line) character, an EOI (End-Or-Identify) asserted in the GPIB interface, or a combination of the two. 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).

New Line Terminator Functions Like EOS and EOT

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

Common Commands within a Subsystem

Common commands can be received and processed by the analyzer whether they are sent over the bus as separate program messages or within other program messages. If you have selected a subsystem, and a common command is received by the analyzer, the analyzer remains in the selected subsystem. For example, if the program message

```
":ACQUIRE:AVERAGE ON;*CLS;COUNT 1024"
```

is received by the analyzer, the analyzer turns averaging on, then clears the status information without leaving the selected subsystem.

If some other type of command is received within a program message, you must re-enter the original subsystem after the command. For example, the program message

```
":ACQUIRE:AVERAGE ON;:AUTOSCALE;:ACQUIRE:AVERAGE:COUNT 1024"
```

turns averaging on, completes the autoscale operation, then sets the acquire average count. In this example, :ACQUIRE must be sent again after the AUTOSCALE command to re-enter the ACQUIRE subsystem and set count.

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 lets you enter a new subsystem. For example:

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

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

You Can Combine Compound and Simple Commands

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

File Names and Types

When specifying a file name in a remote command, enclose the name in double quotation marks, such as "filename". If you specify a path, the path should be included in the quotation marks.

You can use the full path name, a relative path name, or no path. For example, you can specify:

- a full path name: "C:\User Files\waveforms\channel2.wfm"
- a relative path name: "..\myfile.set"
- a simple file name: "Memory1.txt"

All files stored using remote commands have file name extensions.The following table shows the file name extension used for each file type.

Table 1-1. File Name Extensions

File Type	File Name Extension
Waveform - internal format	.wfm
Waveform - text format (Verbose or Y values)	.txt
Setup	.set
Color grade - Gray Scale	.cgs
Screen image	.bmp, .eps, .gif, .pcx, .ps
Mask	.msk
TDR/TDT	.tdr

If you do not specify an extension when storing a file, or specify an incorrect extension, it will be corrected automatically according to the following rules:

- No extension specified: add the extension for the file type.
- Extension does not match file type: Retain the filename and change to the appropriate extension.

You do not need to use an extension when loading a file if you use the optional destination parameter. For example, :DISK:LOAD "STM1_OC3",MASK will automatically add .msk to the file name.

The following table shows the rules used when loading a specified file.

Table 1-2. Rules for Loading Files

File Name Extension	Destination	Rule
No extension	Not specified	Default to internal waveform format; add .wfm extension
Extension does not match file type	Not specified	Default to internal waveform format; add .wfm extension
Extension matches file type	Not specified	Use file name with no alterations; destination is based on extension file type
No extension	Specified	Add extension for destination type; default for waveforms is internal format (.wfm)
Extension does not match destination file type	Specified	Retain file name; add extension for destination type. Default for waveforms is internal format (.wfm)

Table 1-2. Rules for Loading Files (Continued)

File Name Extension	Destination	Rule
Extension matches destination file type	Specified	Retain file name; destination is as specified

Note

ASCII waveform files can be loaded only if the file name explicitly includes the .txt extension.

File Locations

If you don't specify a directory when storing a file, the location of the file will be based on the file type. The following table shows the default locations for storing files.

Table 1-3. Default File Locations (Storing Files)

File Type	Default Location
Waveform - internal format	C:\User Files\waveforms
Waveform - text format (Verbose or Y values)	C:\User Files\waveforms
Setup	C:\User Files\setups
Color Grade - Gray Scale	C:\User Files\colorgrade-grayscale
Screen Image	C:\User Files\screen images
Mask	C:\Scope\masks (for standard masks)
TDR/TDT calibration data	C:\User Files\TDR normalization

When loading a file, you can specify the full path name, a relative path name, or no path name. The following table shows the rules for locating files, based on the path specified.

Table 1-4. File Locations (Loading Files)

File Name	Rule
Full path name	Use file name and path specified
Relative path name	Full path name is formed relative to the present working directory (determine using :DISK:PWD?)
File name with no preceding path	Add the file name to the default path based on the file type.

Files may be stored to or loaded from an internal hard drive under the root path C:\User Files only. The only exceptions are the standard masks loaded from C:\Scope\masks. Attempting to access files outside the root path will generate an error message.

Files may be stored to or loaded from any path on the A:\ drive or on any mapped network drive.

Getting Started Programming

The remainder of this chapter discusses how to set up the analyzer, how to retrieve setup information and measurement results, how to digitize a waveform, and how to pass data to the computer. Chapter 22, “Measure Commands” describes sending measurement data to the analyzer.

Initialization

To make sure the bus and all appropriate interfaces are in a known state, begin every program with an initialization statement. For example, HP BASIC provides a CLEAR command which clears the interface buffer:

```
CLEAR 707 ! initializes the interface of the analyzer
```

When you are using GPIB, CLEAR also resets the analyzer's parser. The parser is the program that reads in the instructions you send.

After clearing the interface, initialize the analyzer to a preset state:

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

Initializing the analyzer

The commands and syntax for initializing the analyzer are discussed in Chapter 7, “Common Commands”. Refer to your GPIB manual and programming language reference manual for information on initializing the interface.

Autoscale

The AUTOSCALE feature of Agilent Technologies digitizing analyzers performs a very useful function on unknown waveforms by automatically setting up the vertical channel, time base, and trigger level of the analyzer.

The syntax for the autoscale function is:

```
:AUTOSCALE<terminator>
```

Setting Up the Analyzer

A typical analyzer setup configures 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 analyzer are:

```
:CHANNEL1:RANGE 16;OFFSET 1.00<terminator>
:SYSTEM:HEADER OFF<terminator>
:TIMEBASE:RANGE 1E-3;DELAY 100E-6<terminator>
```

This example sets the time base at 1 ms full-scale (100 μs/div), with delay of 100 μs. Vertical is set to 16 V full-scale (2 V/div), with center of screen at 1 V, and probe attenuation of 10.

Example Program

This program demonstrates the basic command structure used to program the analyzer.

```
10   CLEAR 707 ! Initialize analyzer interface
20   OUTPUT 707;"*RST" !Initialize analyzer to preset state
30   OUTPUT 707;":TIMEBASE:RANGE 5E-4"! Time base to 500 us full scale
40   OUTPUT 707;":TIMEBASE:DELAY 25E-9"! Delay to 25 ns
50   OUTPUT 707;":TIMEBASE:REFERENCE CENTER"! Display reference at center
60   OUTPUT 707;":CHANNEL1:RANGE .16"! Vertical range to 160 mV full scale
70   OUTPUT 707;":CHANNEL1:OFFSET -.04"! Offset to -40 mV
80   OUTPUT 707;":TRIGGER:LEVEL,-.4"! Trigger level to -0.4
90   OUTPUT 707;":TRIGGER:SLOPE POSITIVE"! Trigger on positive slope
100  OUTPUT 707;":SYSTEM:HEADER OFF"<terminator>
110  OUTPUT 707;":DISPLAY:GRATICULE FRAME"! Grid off
120  END
```

Overview of the Program

- Line 10 initializes the analyzer interface to a known state.
- Line 20 initializes the analyzer to a preset state.
- Lines 30 through 50 set the time base, the horizontal time at 500 µs full scale, and 25 ns of delay referenced at the center of the graticule.
- Lines 60 through 70 set the vertical range to 160 millivolts full scale and the center screen at –40 millivolts.
- Lines 80 through 90 configure the analyzer to trigger at –0.4 volts with normal triggering.
- Line 100 turns system headers off.
- Line 110 turns the grid off.

Using the DIGITIZE Command

The DIGITIZE command is a macro that captures data using the acquisition (ACQUIRE) subsystem. When the digitize process is complete, the acquisition is stopped. The captured data can then be measured by the analyzer or transferred to the computer for further analysis. The captured data consists of two parts: the preamble and the waveform data record.

After changing the analyzer configuration, the waveform buffers are cleared. Before doing a measurement, the DIGITIZE command should be sent to ensure new data has been collected.

You can send the DIGITIZE command with no parameters for a higher throughput. Refer to the DIGITIZE command in Chapter 8, “Root Level Commands” for details.

When the DIGITIZE command is sent to an analyzer, the specified channel’s waveform is digitized with the current ACQUIRE parameters. Before sending the :WAVEFORM:DATA? query to get waveform data, specify the WAVEFORM parameters.

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 contains. The following program example shows a typical setup:

```
OUTPUT 707;":SYSTEM:HEADER OFF"<terminator>
OUTPUT 707;":WAVEFORM:SOURCE CHANNEL1"<terminator>
OUTPUT 707;":WAVEFORM:FORMAT BYTE"<terminator>
OUTPUT 707;":ACQUIRE:COUNT 8"<terminator>
OUTPUT 707;":ACQUIRE:POINTS 500"<terminator>
OUTPUT 707;":DIGITIZE CHANNEL1"<terminator>
OUTPUT 707;":WAVEFORM:DATA?"<terminator>
```

This setup places the analyzer to acquire eight averages. This means that when the DIGITIZE command is received, the command will execute until the waveform has been averaged at least eight times.

After receiving the :WAVEFORM:DATA? query, the analyzer will start passing the waveform information when queried.

Digitized waveforms are passed from the analyzer to the computer 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 BYTE, WORD, or ASCII.

The easiest method of entering a digitized waveform depends on data structures, available formatting, and I/O capabilities. You must scale the integers to determine the voltage value of each point. These integers are passed starting with the leftmost point on the analyzer's display. For more information, refer to Chapter 26, “Waveform Commands”.

When using GPIB, a digitize operation may be aborted by sending a Device Clear over the bus (for example, CLEAR 707).

Receiving Information from the Analyzer

After receiving a query (command header followed by a question mark), the analyzer places the answer in its output queue. The answer remains in the output queue until it is read or until another command is issued. When read, the answer is transmitted across the interface to the computer. The input statement for receiving a response message from an analyzer'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 :CHANNEL1:RANGE? you would execute the HP BASIC statement:

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

The device address parameter represents the address of the analyzer. This would enter the current setting for the range in the string variable Setting$.

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 an input statement. In HP BASIC, this is usually done with an ENTER statement.

Handling Queries Properly

If you send another command or query before reading the result of a query, the output buffer will be cleared and the current response will be lost. This will also generate a query-interrupted error in the error queue. If you execute an input statement before you send a query, it will cause the computer to wait indefinitely.

The format specification for handling response messages depends on both the computer and the programming language.

String Variable Example

The output of the analyzer 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.

For the example programs, assume that the device being programmed is at device address 707. The actual address depends on how you have configured the bus for your own application.

In HP BASIC 5.0, string variables are case-sensitive, and must be expressed exactly the same way each time they are used. This 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 computer displays:

```
+8.00000E-01
```

Numeric Variable Example

This example shows the data being returned to a numeric variable:

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

After running this program, the computer displays:

```
.8
```

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 pound 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 4000 bytes of data, the syntax would be:

```
#44000 <4000 bytes of data> <terminator>
```

The leftmost “4” represents the number of digits in the number of bytes, and “4000” represents the number of bytes to be transmitted.

Multiple Queries

You can send multiple queries to the analyzer 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>
```

Use the following program message to read the query :TIMEBASE:RANGE?;DELAY? into multiple numeric variables:

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

Analyzer Status

Status registers track the current status of the analyzer. By checking the analyzer status, you can find out whether an operation has completed, is receiving triggers, and more. Chapter 4, “Status Reporting” explains how to check the status of the analyzer.

2

Interface Functions

Interface Functions

The interface functions deal with general bus management issues, as well as messages that can be sent over the bus as bus commands. In general, these functions are defined by IEEE 488.1.

GPIB Interface Connector

The analyzer is equipped with a GPIB interface connector on the rear panel. This allows direct connection to a GPIB equipped computer. You can connect an external GPIB compatible device to the analyzer by installing a GPIB cable between the two units. Finger tighten the captive screws on both ends of the GPIB cable to avoid accidentally disconnecting the cable during operation.

A maximum of fifteen GPIB compatible instruments (including a computer) can be interconnected in a system by stacking connectors. This allows the analyzers to be connected in virtually any configuration, as long as there is a path from the computer to every device operating on the bus.

CAUTION Avoid stacking more than three or four cables on any one connector. Multiple connectors produce leverage that can damage a connector mounting.

GPIB Default Startup Conditions

The following default GPIB conditions are established during power-up.

- The Request Service (RQS) bit in the status byte register is set to zero.
- All of the event registers, the Standard Event Status Enable Register, Service Request Enable Register, and the Status Byte Register are cleared.

Interface Capabilities

The interface capabilities of this analyzer, as defined by IEEE 488.1, are listed in the following table.

Table 2-1. Interface Capabilities

Code	Interface Function	Capability
SH1	Source Handshake	Full Capability
AH1	Acceptor Handshake	Full Capability
T5	Talker	Basic Talker/Serial Poll/Talk Only Mode/ Unaddress if Listen Address (MLA)
L4	Listener	Basic Listener/ Unaddresses if Talk Address (MTA)
SR1	Service Request	Full Capability
RL1	Remote Local	Complete Capability
PP1	Parallel Poll	Remote Configuration
DC1	Device Clear	Full Capability
DT1	Device Trigger	Full Capability
C0	Computer	No Capability
E2	Driver Electronics	Tri State (1 MB/SEC MAX)

Command and Data Concepts

The GPIB has two modes of operation, command mode and data mode. The bus is in the command mode when the Attention (ATN) control line is true. The command mode is used to send talk and listen addresses and various bus commands such as 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 analyzer specific commands, queries, and responses found in this manual, including analyzer status information.

Communicating Over the Bus

Device addresses are sent by the computer in the command mode to specify who talks and who listens. Because GPIB can address multiple devices through the same interface card, the device address passed with the program message must include the correct interface select code and the correct analyzer address.

Device Address = (Interface Select Code * 100) + (Analyzer Address)

The Analyzer is at Address 707 in Examples

The examples in this manual assume that the analyzer is at device address 707.

Interface Select Code

Each interface card has a unique interface select code. This code is used by the computer to direct commands and communications to the proper interface. The default is typically “7” for GPIB interface cards.

Analyzer Address

Each analyzer on the GPIB must have a unique analyzer address between decimal 0 and 30. This analyzer address is used by the computer to direct commands and communications to the proper analyzer on an interface. The default is typically “7” for this analyzer. You can change the analyzer address in the Utilities, Remote Interface dialog box.

Do Not Use Address 21 for an Analyzer Address

Address 21 is usually reserved for the Computer interface Talk/Listen address and should not be used as an analyzer address.

Bus Commands

The following commands are IEEE 488.1 bus commands (ATN true). IEEE 488.2 defines many of the actions that are taken when these commands are received by the analyzer.

Device Clear

The device clear (DCL) and selected device clear (SDC) commands clear the input buffer and output queue, reset the parser, and clear any pending commands. If either of these commands is sent during a digitize operation, the digitize operation is aborted.

Group Execute Trigger

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

Interface Clear

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 computer.

3

Message Communication and System Functions

Message Communication and System Functions

This chapter describes the operation of analyzers that operate in compliance with the IEEE 488.2 (syntax) standard. It is intended to give you enough basic information about the IEEE 488.2 standard to successfully program the analyzer. You can find additional detailed information about the IEEE 488.2 standard in ANSI/IEEE Std 488.2-1987, *"IEEE Standard Codes, Formats, Protocols, and Common Commands."*

This analyzer series is designed to be compatible with other Agilent Technologies IEEE 488.2 compatible instruments. Analyzers that are compatible with IEEE 488.2 must also be compatible with IEEE 488.1 (GPIB bus standard); however, IEEE 488.1 compatible analyzers may or may not conform to the IEEE 488.2 standard. The IEEE 488.2 standard defines the message exchange protocols by which the analyzer and the computer will communicate. It also defines some common capabilities that are found in all IEEE 488.2 analyzers.

This chapter also contains some information about the message communication and system functions not specifically defined by IEEE 488.2.

Protocols

The message exchange protocols of IEEE 488.2 define the overall scheme used by the computer and the analyzer 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, you should understand a few system components.

Input Buffer

The input buffer of the analyzer is the memory area where commands and queries are stored prior to being parsed and executed. It allows a computer to send a string of commands, which could take some time to execute, to the analyzer, then proceed to talk to another analyzer while the first analyzer is parsing and executing commands.

Output Queue

The output queue of the analyzer is the memory area where all output data, or response messages, are stored until read by the computer.

Parser

The analyzer's parser is the component that interprets the commands sent to the analyzer and decides what actions should be taken. “Parsing” refers to the action taken by the parser to achieve this goal. Parsing and execution of commands begins when either the analyzer recognizes a program message terminator, or the input buffer becomes full. If you want to send a long sequence of commands to be executed, then talk to another analyzer while they are executing, you should send all of the commands before sending the program message terminator.

Protocol Overview

The analyzer and computer communicate using program messages and response messages. These messages serve as the containers into which sets of program commands or analyzer responses are placed.

A program message is sent by the computer to the analyzer, and a response message is sent from the analyzer to the computer in response to a query message. A query message is defined as being a program message that contains one or more queries. The analyzer will only talk when it has received a valid query message and, therefore, has something to say. The computer should only attempt to read a response after sending a complete query message, but before sending another program message.

Remember This Rule of Analyzer Communication

The basic rule to remember is that the analyzer will only talk when prompted to, and it then expects to talk before being told to do something else.

Protocol Operation

When the analyzer is turned on, the input buffer and output queue are cleared, and the parser is reset to the root level of the command tree.

The analyzer and the computer communicate by exchanging complete program messages and response messages. This means that the computer should always terminate a program message before attempting to read a response. The analyzer will terminate response messages except during a hardcopy output.

After a query message is sent, the next message should be the response message. The computer should always read the complete response message associated with a query message before sending another program message to the same analyzer.

The analyzer allows the computer to send multiple queries in one query message. This is referred to as sending a “compound query”. 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.

Protocol Exceptions

If an error occurs during the information exchange, the exchange may not be completed in a normal manner.

Suffix Multiplier

The suffix multipliers that the analyzer will accept are shown in Table 3-1.

Table 3-1. <suffix mult>

Value	Mnemonic	Value	Mnemonic
1E18	EX	1E-3	m
1E15	PE	1E-6	u
1E12	T	1E-9	n
1E9	G	1E-12	p
1E6	MA	1E-15	f
1E3	K	1E-18	a

Suffix Unit

The suffix units that the analyzer will accept are shown in Table 3-2.

Table 3-2. <suffix unit>

Suffix	Referenced Unit
V	Volt
s	Second

4

Status Reporting

Status Reporting

An overview of the analyzer's status reporting structure is shown in the following figure. The status reporting structure shows you how to monitor specific events in the analyzer. Monitoring these events allows determination of the status of an operation, the availability and reliability of the measured data, and more.

- To monitor an event, first clear the event, then enable the event. All of the events are cleared when you initialize the analyzer.
- To generate a service request (SRQ) interrupt to an external computer, enable at least one bit in the Status Byte Register.

The Status Byte Register, the Standard Event Status Register group, and the Output Queue are defined as the Standard Status Data Structure Model in IEEE 488.2-1987. IEEE 488.2 defines data structures, commands, and common bit definitions for status reporting. There are also analyzer-defined structures and bits.

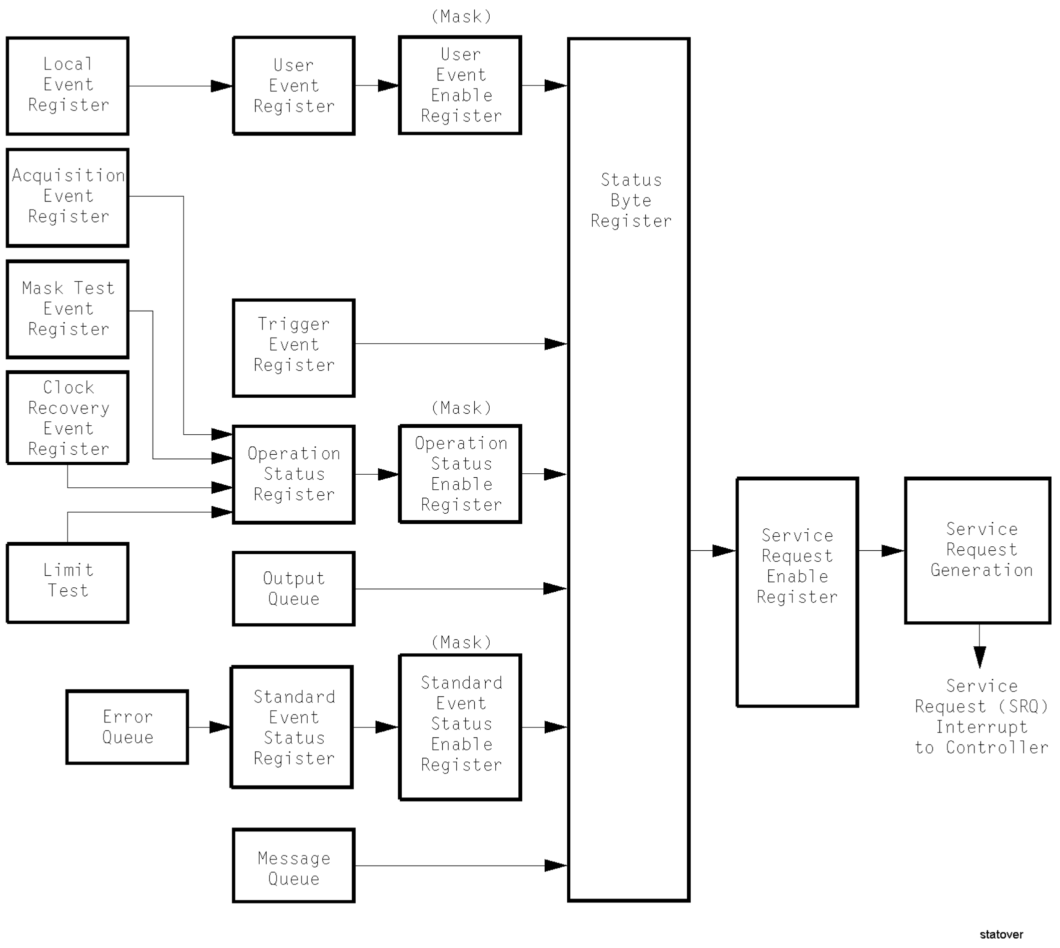

Local Event Register
User Event Register
(Mask)
User Event Enable Register
Acquisition Event Register
Status Byte Register
Mask Test Event Register
Trigger Event Register
Clock Recovery Event Register
(Mask)
Operation Status Register
Operation Status Enable Register
Limit Test
Output Queue
Service Request Enable Register
Service Request Generation
Service Request (SRQ) Interrupt to Controller
(Mask)
Error Queue
Standard Event Status Register
Standard Event Status Enable Register
Message Queue
statover

Figure 4-1. Status Reporting Overview Block Diagram

The status reporting structure consists of the registers shown in this figure.

Status Reporting

The following table lists the bit definitions for each bit in the status reporting data structure.

Table 4-1. Status Reporting Bit Definition

Bit	Description	Definition
PON	Power On	Indicates power is turned on.
URQ		Not used. Permanently set to zero.
CME	Command Error	Indicates if the parser detected an error.
EXE	Execution Error	Indicates if a parameter was out of range or was inconsistent with the current settings.
DDE	Device Dependent Error	Indicates if the device was unable to complete an operation for device dependent reasons.
QYE	Query Error	Indicates if the protocol for queries has been violated.
RQL	Request Control	Indicates if the device is requesting control.
OPC	Operation Complete	Indicates if the device has completed all pending operations.
OPER	Operation Status Register	Indicates if any of the enabled conditions in the Operation Status Register have occurred.
RQS	Request Service	Indicates that the device is requesting service.
MSS	Master Summary Status	Indicates if a device has a reason for requesting service.
ESB	Event Status Bit	Indicates if any of the enabled conditions in the Standard Event Status Register have occurred.
MAV	Message Available	Indicates if there is a response in the output queue.
MSG	Message	Indicates if an advisory has been displayed.
USR	User Event Register	Indicates if any of the enabled conditions have occurred in the User Event Register.
TRG	Trigger	Indicates if a trigger has been received.
LCL	Local	Indicates if a remote-to-local transition occurs.
FAIL	Fail	Indicates the specified test has failed.

Table 4-1. Status Reporting Bit Definition (Continued)

Bit	Description	Definition
COMP	Complete	Indicates the specified test has completed.
LTEST	Limit Test	Indicates that one of the enabled conditions in the Limit Test Register has occurred.
MTEST	Mask Test	Indicates that one of the enabled conditions in the Mask Test Register has occurred.
ACQ	Acquisition	Indicates that acquisition test has completed in the Acquisition Register.
CLCK	CloCk	Indicates that one of the enabled conditions in the Clock Recovery Register has occurred.
UNLK	UNLoCKed	Indicates that an unlocked condition has occurred in the Clock Recovery Module.
LOCK	LOCKed	Indicates that a locked condition has occurred in the Clock Recovery Module.
NSPR1	No Signal Present Receiver 1	Indicates that the Clock Recovery Module has detected the loss of an optical signal on receiver one.
SPR1	Signal Present Receiver 1	Indicates that the Clock Recovery Module has detected an optical signal on receiver one.
NSPR2	No Signal Present Receiver 2	Indicates that the Clock Recovery Module has detected the loss of an optical signal on receiver two.
SPR2	Signal Present Receiver 2	Indicates that the Clock Recovery Module has detected an optical signal on receiver two.

Status Reporting Data Structures

The different status reporting data structures, descriptions, and interactions are shown in the following figure. To make it possible for any of the Standard Event Status Register bits to generate a summary bit, the corresponding bits must be enabled. These bits are enabled by using the *ESE common command to set the corresponding bit in the Standard Event Status Enable Register.

To generate a service request (SRQ) interrupt to the computer, at least one bit in the Status Byte Register must be enabled. These bits are enabled by using the *SRE common command to set the corresponding bit in the Service Request Enable Register. These enabled bits can then set RQS and MSS (bit 6) in the Status Byte Register.

For more information about common commands, see Chapter 7, “Common Commands”.

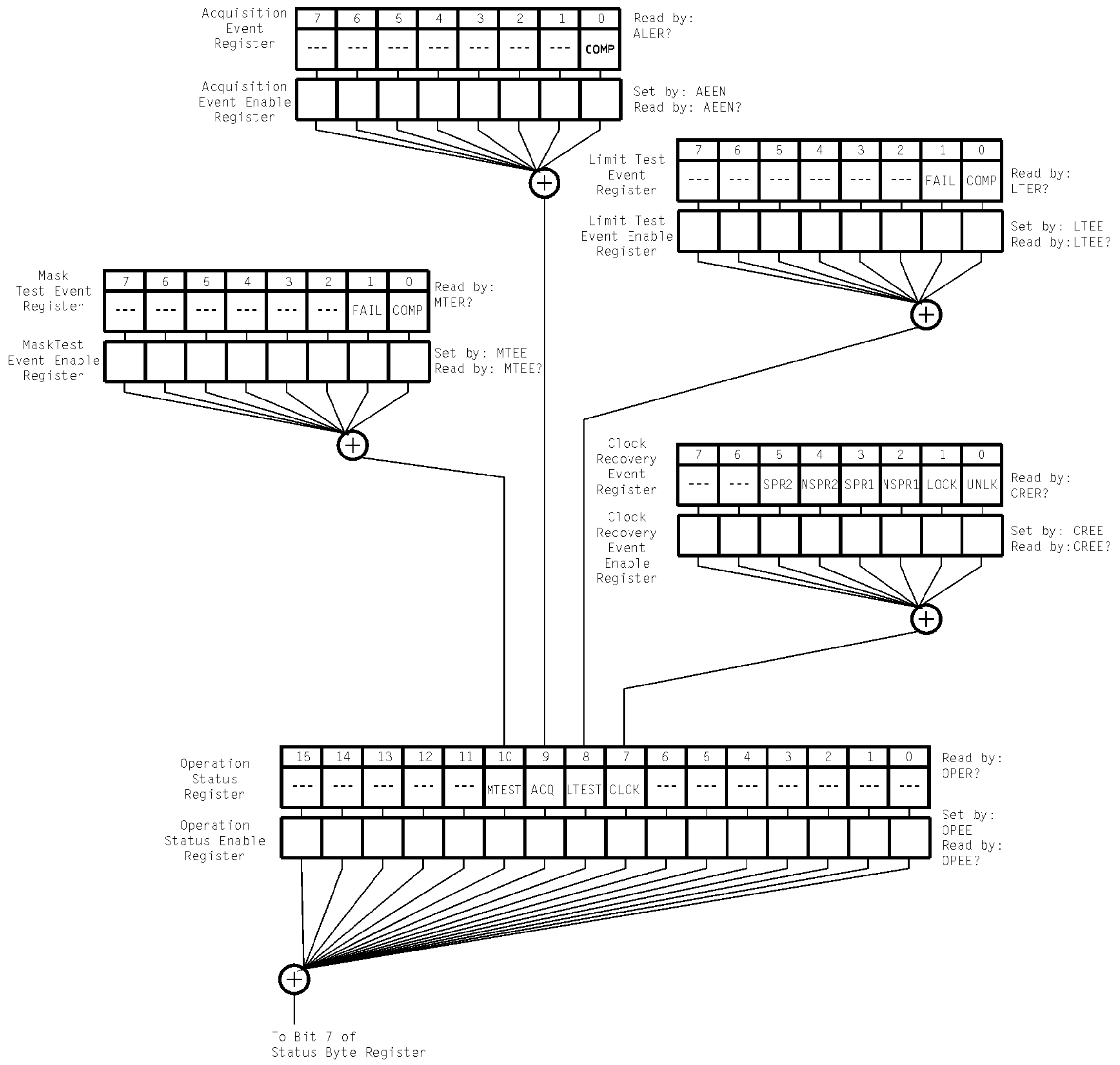
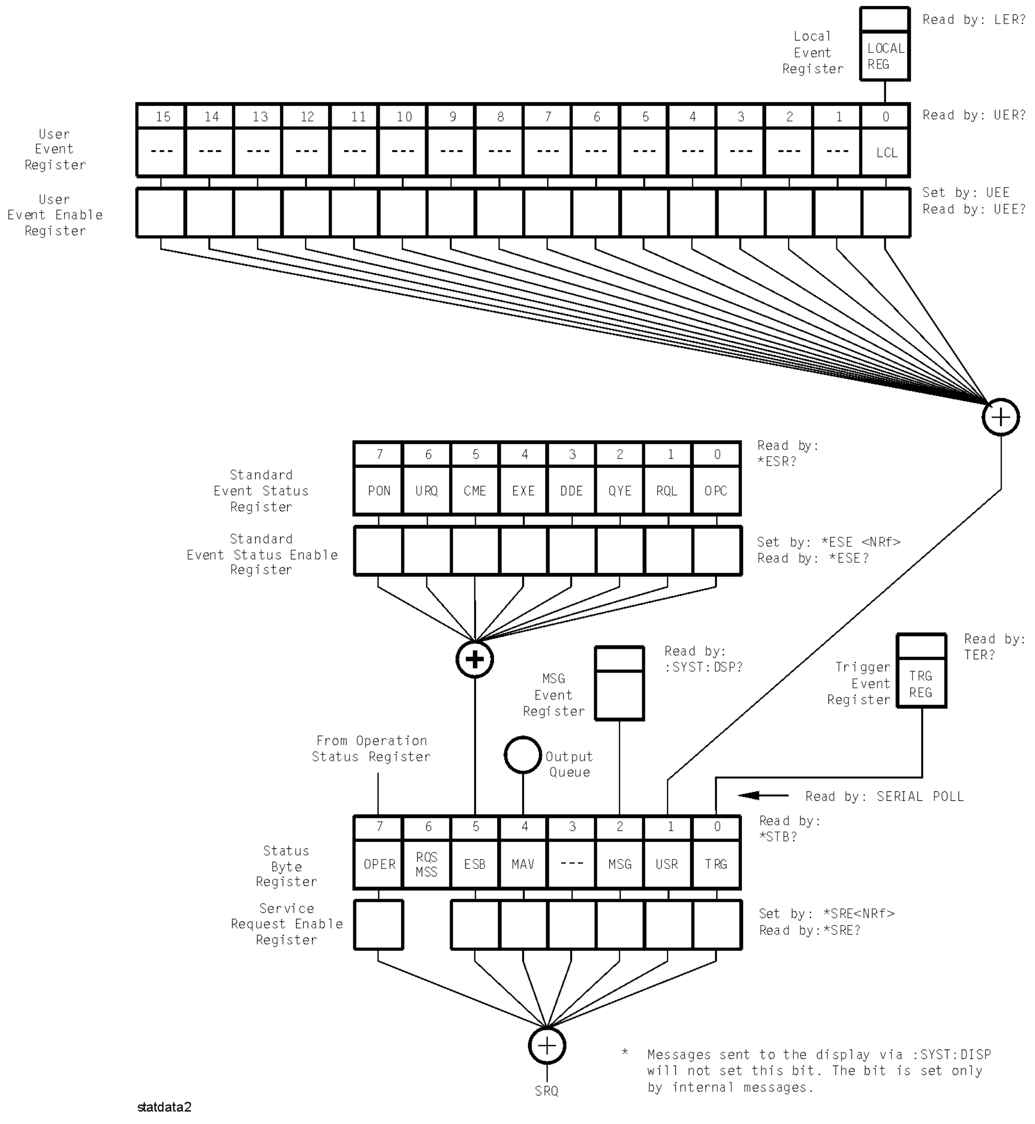

Acquisition Event Register
7 6 5 4 3 2 1 0
--- --- --- --- --- --- --- COMP
Read by: ALER?
Acquisition Event Enable Register
Set by: AEEN
Read by: AEEN?
Limit Test Event Register
7 6 5 4 3 2 1 0
--- --- --- --- --- --- FAIL COMP
Read by: LTER?
Limit Test Event Enable Register
Set by: LTEE
Read by:LTEE?
Mask Test Event Register
7 6 5 4 3 2 1 0
--- --- --- --- --- --- FAIL COMP
Read by: MTER?
MaskTest Event Enable Register
Set by: MTEE
Read by: MTEE?
Clock Recovery Event Register
7 6 5 4 3 2 1 0
--- --- SPR2 NSPR2 SPR1 NSPR1 LOCK UNLK
Read by: CRER?
Clock Recovery Event Enable Register
Set by: CREE
Read by:CREE?
Operation Status Register
15 14 13 12 11 10 9 8 7 6 5 4 3 2 1 0
--- --- --- --- --- MTEST ACQ LTEST CLCK --- --- --- --- --- --- ---
Read by: OPER?
Operation Status Enable Register
Set by: OPEE
Read by: OPEE?
To Bit 7 of Status Byte Register
statdata1

Figure 4-2. Status Reporting Data Structures

Status Reporting Data Structures



Status Reporting Data Structures (continued)

Status Byte Register

The Status Byte Register is the summary-level register in the status reporting structure. It contains summary bits that monitor activity in the other status registers and queues. The Status Byte Register is a live register. That is, its summary bits are set and cleared by the presence and absence of a summary bit from other event registers or queues.

If the Status Byte Register is to be used with the Service Request Enable Register to set bit 6 (RQS/MSS) and to generate an SRQ, at least one of the summary bits must be enabled, then set. Also, event bits in all other status registers must be specifically enabled to generate the summary bit that sets the associated summary bit in the Status Byte Register.

The Status Byte Register can be read using either the *STB? common command query or the GPIB serial poll command. Both commands return the decimal-weighted sum of all set bits in the register. The difference between the two methods is that the serial poll command reads bit 6 as the Request Service (RQS) bit and clears the bit which clears the SRQ interrupt. The *STB? query reads bit 6 as the Master Summary Status (MSS) and does not clear the bit or have any affect on the SRQ interrupt. The value returned is the total bit weights of all of the bits that are set at the present time.

The use of bit 6 can be confusing. This bit was defined to cover all possible computer interfaces, including a computer that could not do a serial poll. The important point to remember is that, if you are using an SRQ interrupt to an external computer, the serial poll command clears bit 6. Clearing bit 6 allows the analyzer to generate another SRQ interrupt when another enabled event occurs.

The only other bit in the Status Byte Register affected by the *STB? query is the Message Available bit (bit 4). If there are no other messages in the Output Queue, bit 4 (MAV) can be cleared as a result of reading the response to the *STB? query.

If bit 4 (weight = 16) and bit 5 (weight = 32) are set, a program would print the sum of the two weights. Since these bits were not enabled to generate an SRQ, bit 6 (weight = 64) is not set.

Example 1

This HP BASIC example uses the *STB? query to read the contents of the analyzer’s Status Byte Register when none of the register's summary bits are enabled to generate an SRQ interrupt.

```
10   OUTPUT 707;":SYSTEM:HEADER OFF;*STB?"!Turn headers off
20   ENTER 707;Result!Place result in a numeric variable
30   PRINT Result!Print the result
40   End
```

The next program prints 132 and clears bit 6 (RQS) of the Status Byte Register. The difference in the decimal value between this example and the previous one is the value of bit 6 (weight = 64). Bit 6 is set when the first enabled summary bit is set, and is cleared when the Status Byte Register is read by the serial poll command.

Example 2

This example uses the HP BASIC serial poll (SPOLL) command to read the contents of the analyzer’s Status Byte Register.

```
10   Result = SPOLL(707)
20   PRINT Result
30   END
```

Use Serial Polling to Read the Status Byte Register

Serial polling is the preferred method to read the contents of the Status Byte Register because it resets bit 6 and allows the next enabled event that occurs to generate a new SRQ interrupt.

Service Request Enable Register

Setting the Service Request Enable Register bits enables corresponding bits in the Status Byte Register. These enabled bits can then set RQS and MSS (bit 6) in the Status Byte Register.

Bits are set in the Service Request Enable Register using the *SRE command, and the bits that are set are read with the *SRE? query. Bit 6 always returns 0. Refer to the Status Reporting Data Structures shown in Figure 4-2.

Example

This example sets bit 4 (MAV) and bit 5 (ESB) in the Service Request Enable Register.

```
OUTPUT 707;"*SRE 48"
```

This example uses the parameter “48” to allow the analyzer to generate an SRQ interrupt under the following conditions:

- When one or more bytes in the Output Queue set bit 4 (MAV).
- When an enabled event in the Standard Event Status Register generates a summary bit that sets bit 5 (ESB).

Trigger Event Register (TRG)

This register sets the TRG bit in the status byte when a trigger event occurs.

The TRG event register stays set until it is cleared by reading the register or using the *CLS (clear status) command. If your application needs to detect multiple triggers, the TRG event register must be cleared after each one.

If you are using the Service Request to interrupt a computer operation when the trigger bit is set, you must clear the event register after each time it is set.

Standard Event Status Register

The Standard Event Status Register (SESR) monitors the following analyzer status events:

- PON - Power On
- CME - Command Error
- EXE - Execution Error
- DDE - Device Dependent Error
- QYE - Query Error
- RQC - Request Control
- OPC - Operation Complete

When one of these events occurs, the corresponding bit is set in the register. If the corresponding bit is also enabled in the Standard Event Status Enable Register, a summary bit (ESB) in the Status Byte Register is set.

The contents of the Standard Event Status Register can be read and the register cleared by sending the *ESR? query. The value returned is the total bit weights of all of the bits set at the present time.

Example

This example uses the *ESR? query to read the contents of the Standard Event Status Register.

```
10   OUTPUT 707;":SYSTEM:HEADER OFF"!Turn headers off
20   OUTPUT 707;"*ESR?"
30   ENTER 707;Result!Place result in a numeric variable
40   PRINT Result!Print the result
50   End
```

If bit 4 (weight = 16) and bit 5 (weight = 32) are set, the program prints the sum of the two weights.

Standard Event Status Enable Register

For any of the Standard Event Status Register (SESR) bits to generate a summary bit, you must first enable the bit. Use the *ESE (Event Status Enable) common command to set the corresponding bit in the Standard Event Status Enable Register. Set bits are read with the *ESE? query.

Example

Suppose your application requires an interrupt whenever any type of error occurs. The error status bits in the Standard Event Status Register are bits 2 through 5. The sum of the decimal weights of these bits is 60. Therefore, you can enable any of these bits to generate the summary bit by sending:

```
OUTPUT 707;"*ESE 60"
```

Whenever an error occurs, the analyzer sets one of these bits in the Standard Event Status Register. Because the bits are all enabled, a summary bit is generated to set bit 5 (ESB) in the Status Byte Register.

If bit 5 (ESB) in the Status Byte Register is enabled (via the *SRE command), a service request interrupt (SRQ) is sent to the external computer.

Disabled SESR Bits Respond, but Do Not Generate a Summary Bit

Standard Event Status Register bits that are not enabled still respond to their corresponding conditions (that is, they are set if the corresponding event occurs). However, because they are not enabled, they do not generate a summary bit in the Status Byte Register.

User Event Register (UER)

This register hosts the LCL bit (bit 0) from the Local Events Register. The other 15 bits are reserved. You can read and clear this register using the UER? query. This register is enabled with the UEE command. For example, if you want to enable the LCL bit, you send a mask value of 1 with the UEE command; otherwise, send a mask value of 0.

Local Event Register (LCL)

This register sets the LCL bit in the User Event Register and the USR bit (bit 1) in the Status byte. It indicates a remote-to-local transition has occurred. The LER? query is used to read and to clear this register.

Operation Status Register (OPR)

This register hosts the CLCK bit (bit 7), the LTEST bit (bit 8), the ACQ bit (bit 9) and the MTEST bit (bit 10).

The CLCK bit is set when any of the enabled conditions in the Clock Recovery Event Register have occurred.

The LTEST bit is set when a limit test fails or is completed and sets the corresponding FAIL or COMP bit in the Limit Test Events Register.

The ACQ bit is set when the COMP bit is set in the Acquisition Event Register, indicating that the data acquisition has satisfied the specified completion criteria.

The MTEST bit is set when the Mask Test either fails specified conditions or satisfies its completion criteria, setting the corresponding FAIl or COMP bits in the Mask Test Events Register.

If any of these bits are set, the OPER bit (bit 7) of the Status Byte register is set. The Operation Status Register is read and cleared with the OPER? query. The register output is enabled or disabled using the mask value supplied with the OPEE command.

Clock Recovery Event Register (CRER)

This register hosts the UNLK bit (bit 0), LOCK bit (bit 1), NSPR1 bit (bit 2), SPR1 bit (bit 3), NSPR2 bit (bit 4) and SPR2 (bit 5).

Bit 0 (UNLK) of the Clock Recovery Event Register is set when Clock Recovery module becomes unlocked.

Bit 1 (LOCK) of the Clock Recovery Event Register is set when Clock Recovery module becomes locked.

Bit 2 (NSPR1) of the Clock Recovery Event Register is set when Clock Recovery module transitions to no longer detecting an optical signal on receiver one.

Bit 3 (SPR1) of the Clock Recovery Event Register is set when Clock Recovery module transitions to detecting an optical signal on receiver one.

Bit 4 (NSPR2) of the Clock Recovery Event Register is set when Clock Recovery module transitions to no longer detecting an optical signal on receiver two.

Bit 5 (SPR2) of the Clock Recovery Event Register is set when Clock Recovery module transitions to detecting an optical signal on receiver two.

The Clock Recovery Event Register is read and cleared with the CRER? query.

When either of the UNLK, LOCK, NSPR1, SPR1, NSPR2 or SPR2 bits are set, they in turn set CLCK bit (bit 7) of the Operation Status Register. Results from the Clock Recovery Event Register can be masked by using the CREE command to set the Clock Recovery Event Enable Register. Refer to the CREE command in Chapter 8, "Root Level Commands" for enable and mask value definitions.

Limit Test Event Register (LTER)

Bit 0 (COMP) of the Limit Test Event Register is set when the Limit Test completes. The Limit Test completion criteria are set by the LTESt:RUN command.

Bit 1 (FAIL) of the Limit Test Event Register is set when the Limit Test fails. Failure criteria for the Limit Test are defined by the LTESt:FAIL command.

The Limit Test Event Register is read and cleared with the LTER? query.

When either the COMP or FAIL bits are set, they in turn set the LTEST bit (bit 8) of the Operation Status Register. You can mask the COMP and FAIL bits, thus preventing them from setting the LTEST bit, by defining a mask using the LTEE command.

Enable	Mask Value
Block COMP and FAIL	0
Enable COMP, block FAIL	1
Enable FAIL, block COMP	2
Enable COMP and FAIL	3

Acquisition Event Register (AER)

Bit 0 (COMP) of the Acquisition Event Register is set when the acquisition limits complete. The Acquisition completion criteria are set by the ACQuire:RUNtil command. The Acquisition Event Register is read and cleared with the ALER? query.

When the COMP bit is set, it in turn sets the ACQ bit (bit 9) of the Operation Status Register. Results from the Acquisition Register can be masked by using the AEEN command to set the Acquisition Event Enable Register to the value 0. You enable the COMP bit by setting the mask value to 1.

Mask Test Event Register (MTER)

Bit 0 (COMP) of the Mask Test Event Register is set when the Mask Test completes. The Mask Test completion criteria are set by the MTESt:RUMode command.

Bit 1 (FAIL) of the Mask Test Event Register is set when the Mask Test fails. This will occur whenever any sample is recorded within any region defined in the mask.

The Mask Test Event Register is read and cleared with the MTER? query.

When either the COMP or FAIL bits are set, they in turn set the MTEST bit (bit 10) of the Operation Status Register. You can mask the COMP and FAIL bits, thus preventing them from setting the MTEST bit, by setting corresponding bits to zero using the MTEE command.

Enable	Mask Value
Block COMP and FAIL	0
Enable COMP, block FAIL	1
Enable FAIL, block COMP	2
Enable COMP and FAIL	3

Error Queue

As errors are detected, they are placed in an error queue. This queue is first in, first out. If the error queue overflows, the last error in the queue is replaced with error –350, “Queue overflow”. Any time the queue overflows, the oldest errors remain in the queue, and the most recent error is discarded. The length of the analyzer's error queue is 30 (29 positions for the error messages, and 1 position for the “Queue overflow” message).

The error queue is read with the SYSTEM:ERROR? query. Executing this query reads and removes the oldest error from the head of the queue, which opens a position at the tail of the queue for a new error. When all the errors have been read from the queue, subsequent error queries return 0, “No error.”

The error queue is cleared when any of the following occurs:

- When the analyzer is powered up.
- When the analyzer receives the *CLS common command.
- When the last item is read from the error queue.

For more information on reading the error queue, refer to the SYSTEM:ERROR? query in Chapter 9, “System Commands”. For a complete list of error messages, refer to Chapter 29, “Error Messages”.

Output Queue

The output queue stores the analyzer-to-computer responses that are generated by certain analyzer commands and queries. The output queue generates the Message Available summary bit when the output queue contains one or more bytes. This summary bit sets the MAV bit (bit 4) in the Status Byte Register. The output queue may be read with the HP BASIC ENTER statement.

Message Queue

The message queue contains the text of the last message written to the advisory line on the screen of the analyzer. The queue is read with the SYSTEM:DSP? query. Note that messages sent with the SYSTem:DSP command do not set the MSG status bit in the Status Byte Register.

Clearing Registers and Queues

The *CLS common command clears all event registers and all queues except the output queue. If *CLS is sent immediately following a program message terminator, the output queue is also cleared.

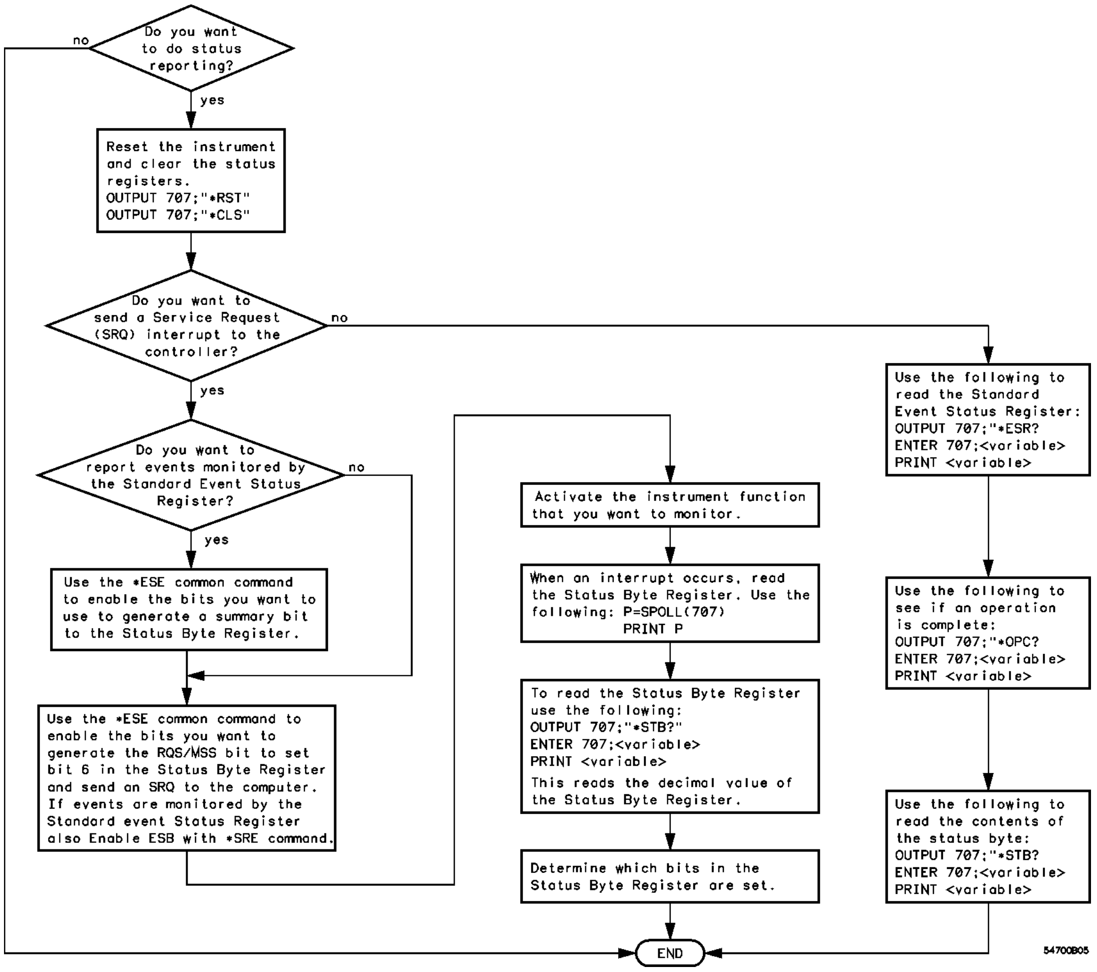

Do you want to do status reporting?
no
yes
Reset the instrument and clear the status registers. OUTPUT 707;"*RST" OUTPUT 707;"*CLS"
Do you want to send a Service Request (SRQ) interrupt to the controller?
no
yes
Do you want to report events monitored by the Standard Event Status Register?
no
yes
Use the *ESE common command to enable the bits you want to use to generate a summary bit to the Status Byte Register.
Use the *ESE common command to enable the bits you want to generate the RQS/MSS bit to set bit 6 in the Status Byte Register and send an SRQ to the computer. If events are monitored by the Standard event Status Register also Enable ESB with *SRE command.
Activate the instrument function that you want to monitor.
When an interrupt occurs, read the Status Byte Register. Use the following: P=SPOLL(707) PRINT P
To read the Status Byte Register use the following: OUTPUT 707;"*STB?" ENTER 707;<variable> PRINT <variable> This reads the decimal value of the Status Byte Register.
Determine which bits in the Status Byte Register are set.
Use the following to read the Standard Event Status Register: OUTPUT 707;"*ESR? ENTER 707;<variable> PRINT <variable>
Use the following to see if an operation is complete: OUTPUT 707;"*OPC? ENTER 707;<variable> PRINT <variable>
Use the following to read the contents of the status byte: OUTPUT 707;"*STB? ENTER 707;<variable> PRINT <variable>
END
54700B05

Figure 4-3. Status Reporting Decision Chart

Programming Conventions

Programming Conventions

This chapter describes conventions used to program the Agilent 86100A, and conventions used throughout this manual. A block diagram and description of data flow is included for understanding analyzer operations. A description of the command tree and command tree traversal is also included. See the Quick Reference for more information about command syntax.

Data Flow

The data flow gives you an idea of where the measurements are made on the acquired data and when the post-signal processing is applied to the data.

The following figure is a block diagram of the analyzer. The diagram is laid out serially for a visual perception of how the data is affected by the analyzer.

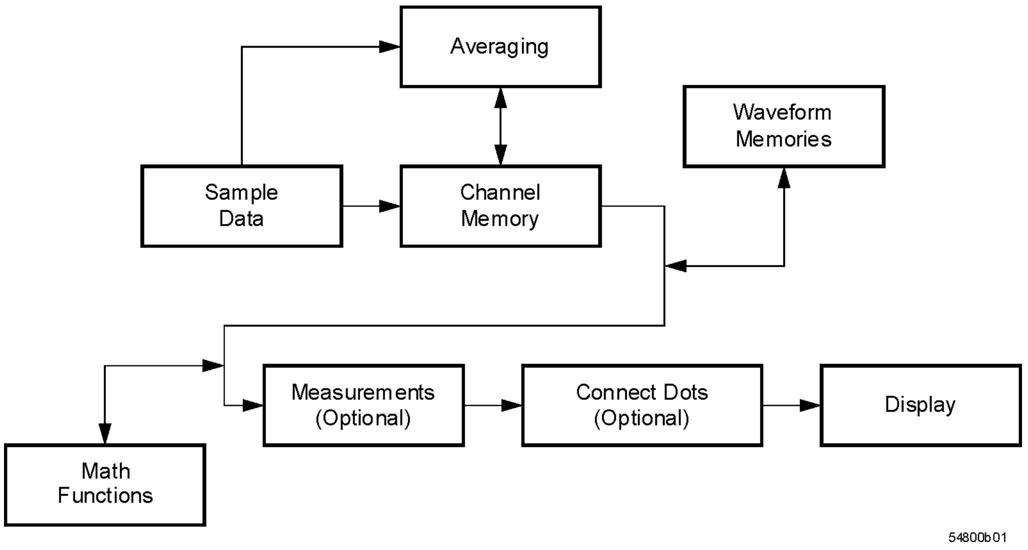


Figure 5-1. Sample Data Processing

The sample data is stored in the channel memory for further processing before being displayed. The time it takes for the sample data to be displayed depends on the number of post processes you have selected.

Averaging your sampled data helps remove any unwanted noise from your waveform.

You can store your sample data in the analyzer's waveform memories for use as one of the sources in Math functions, or to visually compare against a waveform that is captured at a future time. The Math functions allow you to apply mathematical operations on your sampled data. You can use these functions to duplicate many of the mathematical operations that your circuit may be performing to verify that your circuit is operating correctly.

The measurements section performs any of the automated measurements that are available in the analyzer. The measurements that you have selected appear at the bottom of the display.

The Connect Dots section draws a straight line between sample data points, giving an analog look to the waveform. This is sometimes called linear interpolation.

Truncation Rule

The following truncation rule is used to produce the short form (abbreviated spelling) for the mnemonics used in the programming headers and alpha arguments.

Command Truncation Rule

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. If the length of the keyword is four characters or less, this rule does not apply, and the short form is the same as the long form.

The following table shows how the truncation rule is applied to commands.

Table 5-1. Mnemonic Truncation

Long Form	Short Form	How the Rule is Applied
RANGE	RANG	Short form is the first four characters of the keyword.
PATTERN	PATT	Short form is the first four characters of the keyword.
DISK	DISK	Short form is the same as the long form.
DELAY	DEL	Fourth character is a vowel, short form is the first three characters.

The Command Tree

The command tree in Figure 5-2 on page 5-6 shows all of the commands in the Agilent 86100A and the relationship of the commands to each other. The IEEE 488.2 common commands are not listed as part of the command tree because they do not affect the position of the parser within the tree.

When a program message terminator (<NL>, linefeed - ASCII decimal 10) or a leading colon (:) is sent to the analyzer, the parser is set to the “root” of the command tree.

Command Types

The commands in this analyzer can be placed into three types: common commands, root level commands, and subsystem commands.

- Common commands are commands defined by IEEE 488.2 and control some functions that are common to all IEEE 488.2 instruments. These commands are independent of the tree and do not affect the position of the parser within the tree. *RST is an example of a common command.
- Root level commands control many of the basic functions of the analyzer. These commands reside at the root of the command tree. They can always be parsed if they occur at the beginning of a program message or are preceded by a colon. Unlike common commands, root level commands place the parser back at the root of the command tree. AUTOSCALE is an example of a root level command.

- Subsystem commands are grouped together under a common node of the command tree, such as the TIMEBASE commands. Only one subsystem may be selected at a given time. When the analyzer is initially turned on, the command parser is set to the root of the command tree and no subsystem is selected.

See Also
The Quick Reference for information about command syntax and command syntax diagrams.

Tree Traversal Rules

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

Tree Traversal Rules

A leading colon or a program message terminator (<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, use the last mnemonic in the compound header as a reference point (for example, RANGE). Then find the last colon above that mnemonic (TIMEBASE:). That is the point where the parser resides. Any command below this point can be sent within the current program message without sending the mnemonics which appear above them (for example, REFERENCE).

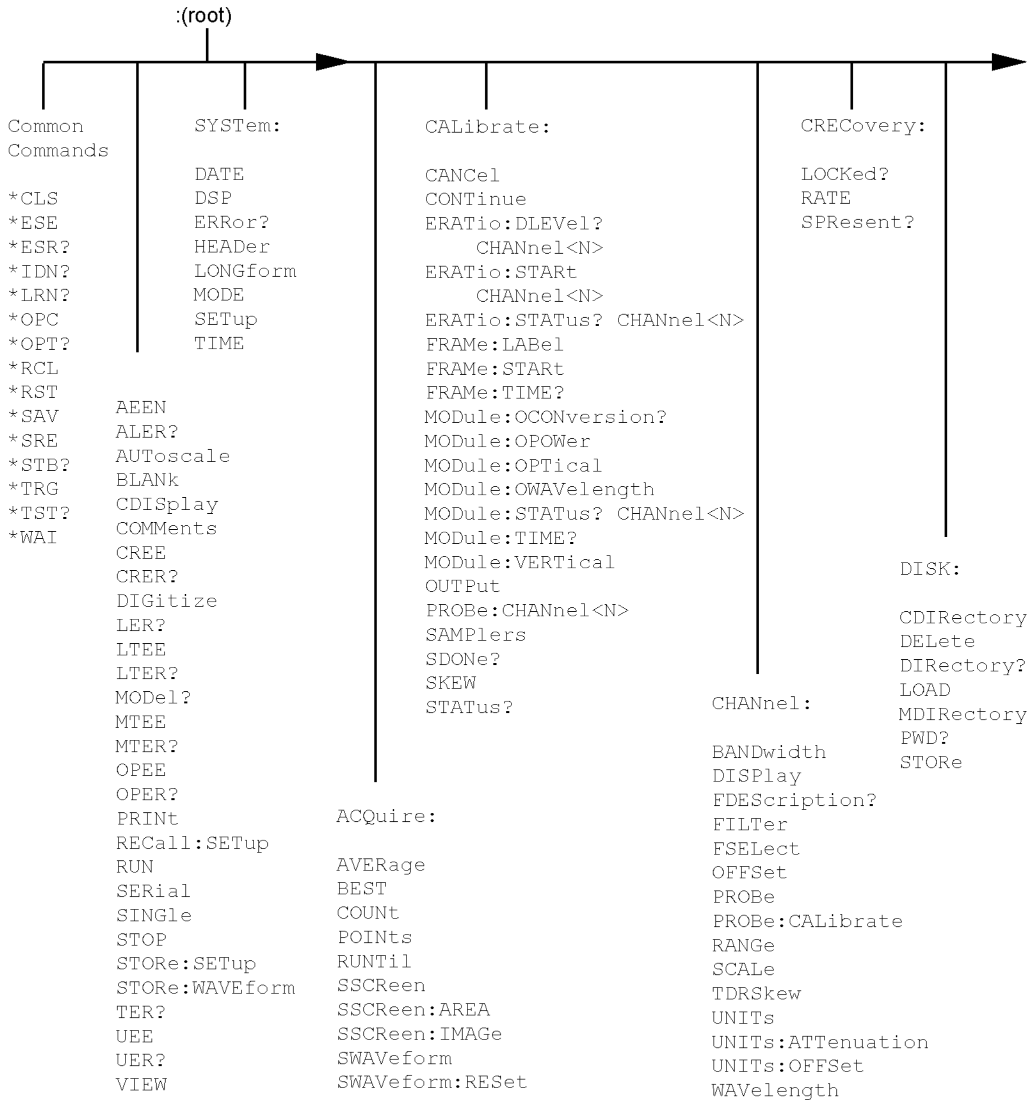
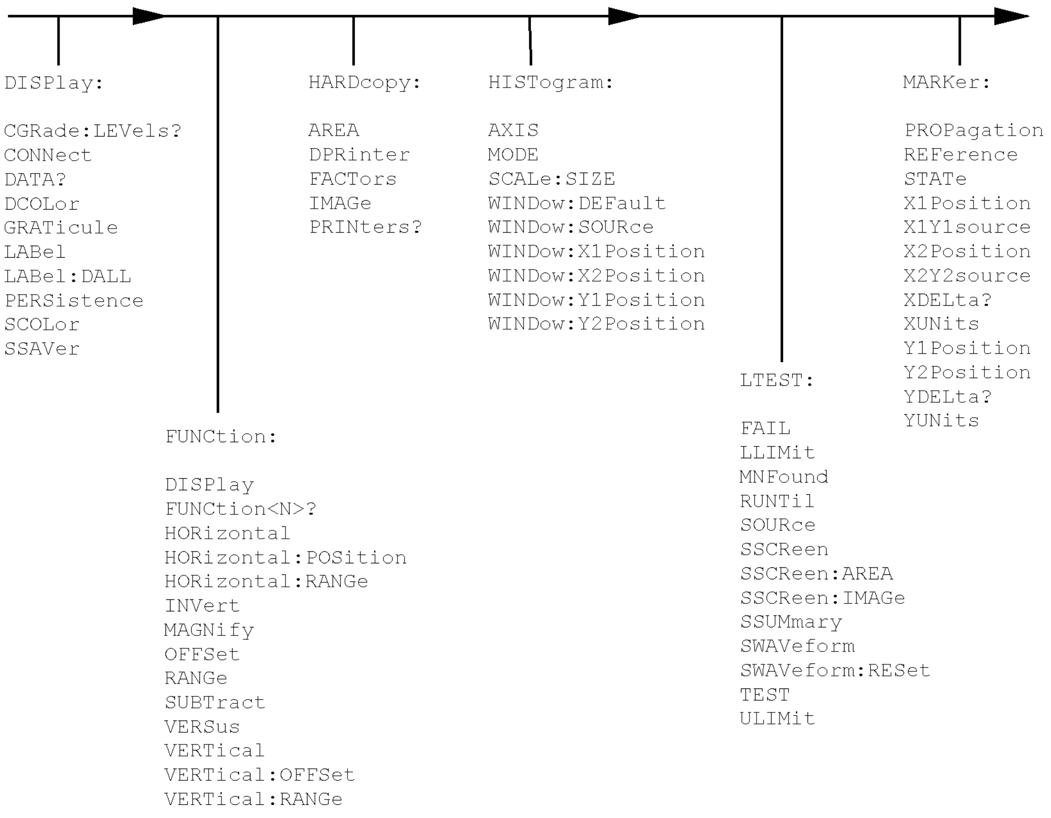


Figure 5-2. Command Tree



```
DISPlay:
CGRade:LEVels?
CONNect
DATA?
DCOLor
GRATicule
LABel
LABel:DALL
PERSistence
SCOLor
SSAVer

FUNCtion:
DISPlay
FUNCtion<N>?
HORizontal
HORizontal:POSition
HORizontal:RANGe
INVert
MAGNify
OFFSet
RANGe
SUBTract
VERSus
VERTical
VERTical:OFFSet
VERTical:RANGe

HARDcopy:
AREA
DPRinter
FACTors
IMAGe
PRINters?

HISTogram:
AXIS
MODE
SCALe:SIZE
WINDow:DEFault
WINDow:SOURce
WINDow:X1Position
WINDow:X2Position
WINDow:Y1Position
WINDow:Y2Position

LTEST:
FAIL
LLIMit
MNFound
RUNTil
SOURce
SSCReen
SSCReen:AREA
SSCReen:IMAGe
SSUMmary
SWAVeform
SWAVeform:RESet
TEST
ULIMit

MARKer:
PROPagation
REFerence
STATe
X1Position
X1Y1source
X2Position
X2Y2source
XDELta?
XUNits
Y1Position
Y2Position
YDELta?
YUNits
```

Command Tree (Continued)

```
MTEST:
ALIGn
AMEThod
COUNt:FAILures?
COUNt:FSAMples?
COUNt:HITS?
COUNt:SAMPles?
COUNt:WAVeforms?
DELete
EXIT
LOAD
MASK:DELete
MMARgin:PERCent
MMARgin:STATe
RUNTil
SCALe:DEFault
SCALe:SOURce?
SCALe:X1
SCALe:XDELta
SCALe:Y1
SCALe:Y2
SCALe:YTRack
SSCReen
SSCReen:AREA
SSCReen:IMAGe
SSUMmary
STARt
SWAVeform
SWAVeform:RESet
TEST
TITLe?
```

```
MEASure:
ANNotation
APOWer
CGRade:AMPLitude
CGRade:BITRate
CGRade:COMPlete
CGRade:CROSsing
CGRade:DCDistortion
CGRade:EHEight
CGRade:ERATio
CGRade:ESN
CGRade:EWIDth
CGRade:JITTer
CGRade:OLEVel
CGRade:PEAK?
CGRade:ZLEVel
CLEar
DEFine
DUTYcycle
FALLtime
FREQuency
HISTogram:HITS
HISTogram:M1S
HISTogram:M2S
HISTogram:M3S
HISTogram:MEAN
HISTogram:MEDian
HISTogram:PEAK
HISTogram:PP
HISTogram:SCALe?
HISTogram:STDDev
NWIDth
OVERshoot
PERiod
PWIDth
RESults?
RISetime
SCRatch
SENDvalid
SOURce
TEDGe?
TVOLt?
VAMPlitude
VBASE
VMAX
VMIN
VPP
VRMS
VTIMe?
VTOP
```

```
TDR:
PRESet
RATE
RESPonse
RESPonse:CALibrate
RESPonse:CALibrate:CANCel
RESPonse:CALibrate:CONTinue
RESPonse:HORizontal
RESPonse:HORizontal:POSition
RESPonse:HORizontal:RANGe
RESPonse:RISetime
RESPonse:TDRDest
RESPonse:TDRTDT
RESPonse:TDTDest
RESPonse:VERTical
RESPonse:VERTical:OFFSet
RESPonse:VERTical:RANGe
STIMulus
```

```
TIMebase:
BRATe
POSition
RANGe
REFerence
SCALe
UNITs
```

```
TRIGger:
ATTenuation
BWLimit
GATed
HYSTeresis
LEVel
SLOPe
SOURce
```

```
WAVeform:
BANDpass?
BYTeorder
COUNt?
DATA
FORMat
POINts?
PREamble
SOURce
TYPE?
XDISplay?
XINCrement?
XORigin?
XRANge?
XREFerence?
XUNits?
YDISplay?
YINCrement?
YORigin?
YRANge?
YREFerence?
YUNits?
```

```
WMEMory:
DISPlay
LOAD
SAVE
XOFFset
XRANge
YOFFset
YRANge
```

Command Tree (Continued)

Tree Traversal Examples

The OUTPUT statements in the following examples are written using HP BASIC 5.0. The quoted string is placed on the bus, followed by a carriage return and linefeed (CRLF).

Example 1

Consider the following command:

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

The colon between CHANNEL1 and RANGE is necessary because CHANNEL1:RANGE is a compound command. The semicolon between the RANGE command and the OFFSET command is required to separate the two commands or operations. The OFFSET command does not need CHANNEL1 preceding it because the CHANNEL1:RANGE command sets the parser to the CHANNEL1 node in the tree.

Example 2

Consider the following commands:

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

or

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

In the first line of example 2, the “subsystem selector” is implied for the POSITION command in the compound command.

A second way to send these commands is shown in the second part of the example. Since the program message terminator places the parser back at the root of the command tree, TIMEBASE must be reselected to re-enter the TIMEBASE node before sending the POSITION command.

Example 3

Consider the following command:

```
OUTPUT 707;":TIMEBASE:REFERENCE CENTER;:CHANNEL1:OFFSET 0"
```

In example 3, the leading colon before CHANNEL1 tells the parser to go back to the root of the command tree. The parser can then recognize the CHANNEL1:OFFSET command and enter the correct node.

Infinity Representation

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

Sequential and Overlapped Commands

IEEE 488.2 makes a distinction between sequential and overlapped commands.

Sequential commands finish their task before the execution of the next command starts.

Overlapped commands run concurrently. Commands following an overlapped command may be started before the overlapped command is completed. The common commands *WAI and *OPC may be used to ensure that commands are completely processed before subsequent commands are executed.

Response Generation

As defined by IEEE 488.2, query responses may be buffered for the following reasons:

- When the query is parsed by the analyzer.
- When the computer addresses the analyzer to talk so that it may read the response.

This analyzer buffers responses to a query when the query is parsed.

EOI

The EOI bus control line follows the IEEE 488.2 standard without exception.

Sample Programs

Sample Programs

Sample programs for the Agilent 86100A analyzers are shipped on a disk with the instrument. Each program demonstrates specific sets of instructions. This chapter shows you some of those functions, and describes the commands being executed. Both C and HP BASIC examples are included.

The header file is:

- hpibdecl.h

The C examples include:

- init.c
- gen_srq.c
- srq.c
- learnstr.c
- sicl_IO.c
- natl_IO.c

The HP BASIC examples include:

- init.bas
- srq.bas
- lrn_str.bas

The sample program listings are included at the end of this chapter.

Sample Program Structure

This chapter includes segments of both the C and HP BASIC sample programs. Each program includes the basic functions of initializing the interface and analyzer, capturing the data, and analyzing the data.

In general, both the C and HP BASIC sample programs typically contain the following fundamental segments:

Segment	Description
main program	Defines global variables and constants, specifies include files, and calls various functions.
initialize	Initializes the GPIB and analyzer, and sets up the analyzer and the ACQuire subsystem.
acquire_data	Digitizes the waveform to capture data.
auto_measurements	Performs simple parametric measurements.
transfer_data	Brings waveform data and voltage/timing information (the preamble) into the computer.

Sample C Programs

Segments of the sample programs “init.c” and “gen_srq.c” are shown and described in this chapter.

init.c - Initialization

```
/*    init. c */

/*    Command Order Example.  This program demonstrates the order of commands suggested for operation of the analyzer via HPIB.
      This program initializes the scope, acquires data, performs automatic measurements, and transfers and stores the data on the PC
      as time/voltage pairs in a comma-separated file format useful for spreadsheet applications. It assumes a SICL INTERFACE exists
      as 'hpib7' and an Agilent 86100A analyzer at address 7. It also requires the cal signal attached to Channel 1.

      See the README file on the demo disk for development and linking information.
*/

# include <stdio.h>                          /* location of: printf ( ) */
# include <stdlib.h>                         /* location of: atof(), atoi ( ) */
# include "hpibdecl.h"                       /* prototypes, global declarations, constants */

void initialize ( );                         /* initialize the scope */
void acquire_data ( );                       /* digitize signal */
void auto_measurements ( );                  /* perform built-in automatic measurements */
void transfer_data ( );                      /* transfers waveform data from scope to PC */
void convert_data ( );                       /* converts data to time/voltage values */
void store_csv ( );                          /* stores time/voltage pairs to comma-separated
                                             /* variable file format */
```

The include statements start the program. The file “hpibdecl.h” includes prototypes and declarations that are necessary for the analyzer sample programs.

This segment of the sample program defines the functions, in order, that are used to initialize the scope, digitize the data, perform measurements, transfer data from the scope to the PC, convert the digitized data to time and voltage pairs, and store the converted data in comma-separated variable file format.

See the following descriptions of the program segments.

init.c - Global Definitions and Main Program

```
/* GLOBALS */
int count;
double xorg,xref,xinc;                    /* values necessary for conversion of data */
double yorg,yref,yinc;
int Acquired_length;
char data[MAX_LENGTH];                    /* data buffer */
double time_value[MAX_LENGTH];            /* time value of data */
double volts[MAX_LENGTH];                 /* voltage value of data */


void main( void )
{
/* initialize interface and device sessions */
/* note: routine found in sicl_IO.c or natl_IO.c  */

    init_IO ( );

    initialize ( );                       /* initialize the scope and interface and set up SRQ  */
    acquire_data ( );                     /* capture the data */
    auto_measurements ( );                /* perform automated measurements on acquired data */
    transfer_data ( );                    /* transfer waveform data to the PC from scope */
    convert_data ( );                     /* convert data to time/voltage pairs */
    store_csv ( );                        /* store the time/voltage pairs as csv file */
    close_IO ( );                         /* close interface and device sessions */
                                          /* note: routine found in sicl_IO.c or natl_IO.c  */

} /* end main ( ) */
```

The init_IO routine initializes the analyzer and interface so that the scope can capture data and perform measurements on the data. At the start of the program, global symbols are defined which will be used to store and convert the digitized data to time and voltage values.

init.c - Initializing the Analyzer

```
/*
*  Function name:  initialize
*  Parameters:  none
*  Return value:  none
*  Description:  This routine initializes the analyzer for proper
*  acquisition of data. The instrument is reset to a known state and the
*  interface is cleared. System headers are turned off to allow faster
*  throughput and immediate access to the data values requested by queries.
*  The analyzer time base, channel, and trigger subsystems are then
*  configured. Finally, the acquisition subsystem is initialized.
*/
void initialize (  )
{
        write_IO ("*RST");                          /* reset scope - initialize to known state */
        write_IO ("*CLS");                          /* clear status registers and output queue */

        write_IO (":SYSTem:HEADer OFF");            /* turn off system headers */

        /* initialize time base parameters to center reference, */
        /* 2 ms full-scale (200 us/div), and 20 us delay */
        write_IO (":TIMebase:REFerence CENTer;RANGe 2e-3;POSition 20e-6");


        /* initialize Channel1 1.6V full-scale (200 mv/div); offset -400mv */
        write_IO (":CHANnel1:RANGe 1.6;OFFSet -400e-3");


        /* initialize trigger info: channel1 signal on positive slope at 300mv */
        write_IO (":TRIGger:SOURce FPANel;SLOPe POSitive");
        write_IO (":TRIGger:LEVel-0.40");


        /* initialize acquisition subsystem */
        /* Real time acquisition - no averaging; record length 4096 */
        write_IO (":ACQuire:AVERage OFF;POINts 4096");

}  /* end initialize ( ) */
```

init.c - Acquiring Data

```
/*
*  Function name:  acquire_data
*  Parameters: none
*  Return value:  none
*  Description:  This routine acquires data according to the current
*  instrument settings.
*/
void acquire_data ( )
{
/*
*  The root level :DIGitize command is recommended for acquisition of new
*  data. It will initialize data buffers, acquire new data, and ensure that
*  acquisition criteria are met before acquisition of data is stopped.  The
*  captured data is then available for measurements, storage, or transfer
*  to a PC.  Note that the display is automatically turned off by the
*  :DIGitize command and must be turned on to view the captured data.
*/

        write_IO (":DIGitize CHANnel1");
        write_IO (":CHANnel1:DISPlay ON");                    /* turn on channel 1 display which is */
                                                              /* turned off by the :DIGitize command */

} /*  end acquire_data ( ) */
```

init.c - Making Automatic Measurements

```
/*
*  Function name:  auto_measurements
*  Parameters:  none
*  Return value:  none
*  Description:  This routine performs automatic measurements of volts
*  peak-to-peak and period on the acquired data. It also demonstrates
*  two methods of error detection when using automatic measurements.
*/

void auto_measurements ( )
{
float period, vpp;
unsigned char vpp_str[16];
unsigned char period_str[16];
int bytes_read;

/*
*  Error checking on automatic measurements can be done using one of two methods.
*  The first method requires that you turn on results in the Measurements
*  subsystem using the command :MEASure:SEND ON.  When this is on, the analyzer
*  will return the measurement and a result indicator.  The result flag is zero
*  if the measurement was successfully completed, otherwise a non-zero value is
*  returned which indicates why the measurement failed. See the Programmer's Manual
*  for descriptions of result indicators.
*
*  The second method simply requires that you check the return value of the
*  measurement.  Any measurement not made successfully will return with the value
*  +9.999E37. This could indicate that either the measurement was unable to be
*  performed, or that insufficient waveform data was available to make the
*  measurement.
*/
/*
*  METHOD ONE - turn on results to indicate whether the measurement completed
*  successfully.  Note that this requires transmission of extra data from the scope.
*/
write_IO (":MEASure:SEND ON");                          /* turn results on */
write_IO (":MEASure:VPP? CHANnel1");                    /* query -- volts peak-to-peak channel 1*/

bytes_read = read_IO(vpp_str,16L);                      /* read in value and result flag */

if (vpp_str[bytes_read-2] != '0')
   printf ("Automated vpp measurement error with result %c\n",
            vpp_str [bytes_read-2]);
else
   printf ("VPP is %f\n", (float) atof (vpp_str));

write_IO (":MEASure:PERiod? CHANnel1");                 /* period channel 1 */

bytes_read = read_IO (period_str,16L);                  /* read in value and result flag */

if period_str[bytes_read-2] != '0')
   printf ("Automated period measurement error with result %c\n",
        period_str [bytes_read-2]);
else
  printf ("Period is %f\n",(float)atof (period_str));

/*
*  METHOD TWO - perform automated measurements and error checking with
*  :MEAS:RESULTS OFF
*/
period = (float) 0;
vpp = (float) 0;

/* turn off results */
write_IO (":MEASure:SEND OFF");

write_IO (":MEASure:PERiod? CHANnel1");            /*period 1 */
bytes_read = read_IO (period_str,16L);             /* read in value and result flag */

period = (float) atof (period_str);

if (period > 9.99e37 )
  printf ("\nPeriod could not be measured.\n");
else
  printf ("\nThe period of channel 1 is %f seconds.\n", period );

write_IO (":MEASure:VPP? CHANnel1");
bytes_read = read_IO ( vpp_str,16L );

vpp = (float) atof (vpp_str);

if ( vpp > 9.99e37 )
  printf ("Peak-to-peak voltage could not be measured.\n");
else
  printf ("The voltage peak-to-peak is %f volts.\n", vpp );

} /* end auto_measurements () */
```

init.c - Error Checking

```
/* Error checking on automatic measurements can be done using one of two methods.
*  The first method requires that you turn on results in the Measurements
*  subsystem using the command :MEASure:SEND ON.  When this is on, the analyzer
*  will return the measurement and a result indicator.  The result flag is zero
*  if the measurement was successfully completed, otherwise a non-zero value is
*  returned which indicates why the measurement failed. See the Programmer's Manual
*  for descriptions of result indicators.

*  The second method simply requires that you check the return value of the
*  measurement. Any measurement not made successfully will return with the value
*  +9.999E37. This could indicate that either the measurement was unable to be
*  performed, or that insufficient waveform data was available to make the
*  measurement.


*  METHOD ONE - turn on results to indicate whether the measurement completed
*  successfully. Note that this requires transmission of extra data from the scope.
*/
        write_IO (":MEASure:SEND ON");                         /* turn results on */

        /* query -- volts peak-to-peak channel 1*/
        write_IO (":MEASure:VPP? CHANnel1");

        bytes_read = read_IO(vpp_str,16L);                     /* read in value and result flag */

        if (vpp_str[bytes_read-2] != '0')
          printf ("Automated vpp measurement error with result %c\n",
          vpp_str[bytes_read-2]);
        else
          printf ("VPP is %f\n",(float)atof(vpp_str));

        write_IO (":MEASure:PERiod? CHANnel1");                /* period channel 1 */
        bytes_read = read_IO(period_str,16L);                  /* read in value and result flag */

        if period_str[bytes_read-2] != '0')
          printf ("Automated period measurement error with result %c\n",
          period_str[bytes_read-2]);
        else
          printf ("Period is %f\n",(float)atof (period_str));
/*
*  METHOD TWO - perform automated measurements and error checking with
*  :MEAS:RESULTS OFF.
*/
period = (float) 0;
vpp = (float) 0;

        /* turn off results */
        write_IO (":MEASure:SEND OFF");

        write_IO (":MEASure:PERiod? CHANnel1");                /* period channel 1 */
        bytes_read = read_IO (period_str,16L);                 /* read in value and result flag */
        period = (float) atof (period_str);

        if ( period > 9.99e37 )
          printf ("\nPeriod could not be measured.\n");
        else
          printf ("\nThe period of channel 1 is %f seconds.\n", period );

        write_IO (":MEASure:VPP? CHANnel1");
        bytes_read = read_IO ( vpp_str,16L );

        vpp = (float) atof (vpp_str);

        if ( vpp > 9.99e37 )
          printf ("Peak-to-peak voltage could not be measured.\n");
        else
          printf ("The voltage peak-to-peak is %f volts.\n", vpp );

} /* end auto_measurements() */
```

init.c - Transferring Data to the PC

```
/*
*  Function name:  transfer_data
*  Parameters:  none
*  Return value:  none
*  Description:  This routine transfers the waveform conversion factors and
*  waveform data to the PC.
*/

void transfer_data ( )
{
        int header_length;
        char header_str[8];
        char term;

        char xinc_str[32],xorg_str[32],xref_str[32];
        char yinc_str[32],yref_str[32],yorg_str[32];

        int bytes_read;

        /* waveform data source channel 1 */
        write_IO (":WAVeform:SOURce CHANnel1");
        /* setup transfer format */
        write_IO (":WAVeform:FORMat BYTE");
        /* request values to allow interpretation of raw data */
        write_IO (":WAVeform:XINCrement?");
        bytes_read = read_IO (xinc_str,32L);
        xinc = atof (xinc_str);

        write_IO (":WAVeform:XORigin?");
        bytes_read = read_IO (xorg_str,32L);
        xorg = atof (xorg_str);

        write_IO (":WAVeform:XREFerence?");
        bytes_read = read_IO (xref_str,32L);
        xref = atof (xref_str);

        write_IO (":WAVeform:YINCrement?");
        bytes_read = read_IO (yinc_str,32L);
        yinc = atof (yinc_str);

        write_IO (":WAVeform:YORigin?");
        bytes_read = read_IO (yorg_str,32L);
        yorg = atof (yorg_str);

        write_IO (":WAVeform:YREFerence?");
        bytes_read = read_IO (yref_str,32L);
        yref = atof (yref_str);

        write_IO (":WAVeform:DATA?");                /* request waveform data */
        while (data[0] != '#')
          bytes_read = read_IO (data,1L);            /* find the # character */
          bytes_read = read_IO (header_str,1L);      /* input byte counter */
        header_length = atoi (header_str);

        /* read number of points - value in bytes */
        bytes_read = read_IO (header_str,(long)header_length);

        Acquired_length = atoi (header_str);                /* number of bytes */

        bytes_read = read_IO (data,Acquired_length);        /* input waveform data */
        bytes_read = read_IO (&term,1L);                    /* input termination character */

} /* end transfer_data () */
```

An example header resembles the following when the information is stripped off:

```
#510225
```

The left-most “5” defines the number of digits that follow (10225). The number “10225” is the number of points in the waveform. The information is stripped off of the header to get the number of data bytes that need to be read from the analyzer.

init.c - Converting Waveform Data

```
/*
*  Function name:  convert_data
*  Parameters:  none
*  Return value:  none
*  Description:  This routine converts the waveform data to time/voltage
*  information using the values that describe the waveform.  These values are
*  stored in global arrays for use by other routines.
*/

void convert_data ( )
{
        int i;

        for (i = 0; i < Acquired_length; i++)
        {
          time_value[i] = ((i - xref) * xinc) + xorg;/* calculate time info */
          volts[i] = ((data[i] - yref) * yinc) + yorg;/* calculate volt info */
        }
} /* end convert_data ( ) */
```

The data values are returned as digitized samples (sometimes called quantization levels or q-levels). These data values must be converted into voltage and time values.

init.c - Storing Waveform Time and Voltage Information

```
/*
*  Function name:  store_csv
*  Parameters:  none
*  Return value:  none
*  Description:  This routine stores the time and voltage information about
*  the waveform as time/voltage pairs in a comma-separated variable file
*  format.
*/

void store_csv ( )
{
        FILE *fp;
        int i;

        fp = fopen ("pairs.csv","wb");                  /* open file in binary mode - clear file */
                                                        /* if already exists */
        if (fp != NULL)
        {
          for (i = 0; i < Acquired_length; i++)
        {
            /* write time,volt pairs to file */
              fprintf ( fp,"%e,%lf\n",time_value[i],volts[i]);
        }
          fclose ( fp );                                /* close file */
        }
        else
          printf ("Unable to open file 'pairs.csv'\n");

}  /* end store_csv ( ) */
```

The time and voltage information of the waveform is stored in integer format, with the time stored first, followed by a comma, and the voltage stored second.

gen_srq.c - Generating a Service Request

Segments of the sample C program “gen_srq.c” show how to initialize the interface and analyzer, and generate a service request.

Two include statements start the “gen_srq.c” program. The file “stdio.h” defines the standard location of the printf routine, and is needed whenever input or output functions are used. The file “hpibdecl.h” includes necessary prototypes and declarations for the analyzers sample programs. The path of these files must specify the disk drive and directory where the “include” files reside.

```
/* gen_srq.c */

/*
*  This example program initializes the Agilent 86100A scope, runs an autoscale,
*  then generates and responds to a Service Request from the scope. The program
* assumes an Agilent 86100A at address 7, an interface card at interface select code 7,
* and a signal source attached to channel 1.
*/

#include <stdio.h>                    /* location of: printf ( ) */
#include "hpibdecl.h"

void initialize ( );
void setup_SRQ ( );
void create_SRQ ( );

void main ( void )
{
        init_IO ( );                  /* initialize interface and device sessions */
        initialize ( );               /* initialize the scope and interface */
        setup_SRQ ( );                /* enable SRQs on scope and set up SRQ handler */
        create_SRQ ( );               /* generate SRQ */
        close_IO ( );                 /* close interface and device sessions */

} /* end main ( ) */
```

The routine “init_IO” contains three subroutines that initialize the analyzer and interface, and sets up and generate a service request.

The following segment describes the initialize subroutine.

Initializing the Analyzer

The following function is demonstrated in the “gen_srq.c” sample program.

```
/*
* Function name:  initialize
* Parameters:  none
* Return value:  none
* Description:  This routine initializes the analyzer for proper acquisition
* of data. The instrument is reset to a known state and the interface is
* cleared. System headers are turned off to allow faster throughput and
* immediate access to the data values requested by queries. The analyzer
* performs an autoscale to acquire waveform data.
*/

void initialize ( )
{
        write_IO ("*RST");                    /* reset scope - initialize to known state */
        write_IO ("*CLS");                    /* clear status registers and output queue */
        write_IO (":SYSTem:HEADer OFF");/* turn off system headers */
        write_IO (":AUToscale");              /* perform autoscale */

} /* end initialize ( ) */
```

The *RST command is a common command that resets the analyzer to a known default configuration. Using this command ensures that the analyzer is in a known state before you configure it. *RST ensures very consistent and repeatable results. Without *RST, a program may run one time, but it may give different results in following runs if the analyzer is configured differently.

For example, if the trigger mode is normally set to edge, the program may function properly. But, if someone puts the analyzer in the advanced TV trigger mode from the front panel, the program may read measurement results that are totally incorrect. So, *RST defaults the scope to a set configuration so that the program can proceed from the same state each time.

The *CLS command clears the status registers and the output queue.

AUToscale finds and displays all signals that are attached to the analyzer. You should program the analyzer’s time base, channel, and trigger for the specific measurement to be made, as you would do from the front panel, and use whatever other commands are needed to configure the analyzer for the desired measurement.

Setting Up a Service Request

The following code segment shows how to generate a service request. The following function is demonstrated in the “gen_srq.c” sample program.

```
/*
*  Function name:  setup_SRQ
*  Parameters:  none
*  Return value:  none
*  Description:  This routine initializes the device to generate Service Requests. It
*  sets the Service Request Enable Register Event Status Bit and the Standard
*  Event Status Enable Register to allow SRQs on Command, Execution, Device
*  Dependent, or Query errors.
*/
void setup_SRQ (  )
{
        /* Enable Service Request Enable Register - Event Status Bit */

        write_IO ("*SRE 32");          /* Enable Standard Event Status Enable Register */
                                       /* enable Command Error - bit 5 - value 32 */
                                       /* Query Error - bit 2 - value 4 */
        write_IO ("*ESE 36");

} /* end setup_SRQ ( ) */
```

Generating a Service Request

The following function is demonstrated in the “gen_srq.c” sample program.

```
/*
*  Function name:  create_SRQ
*  Parameters:  none
*  Return value:  none
*  Description:  This routine sends two illegal commands to the scope which will
*  generate an SRQ and will place two error strings in the error queue.  The scope
*  ID is requested to allow time for the SRQ to be generated.  The ID string
*  will contain a leading character which is the response placed in the output
*  queue by the interrupted query.
*/

void create_SRQ ( )
{
        char buf [256] = { 0 };  //read buffer for id string
        int bytes_read = 0;
        int srq_asserted;

        /* Generate query error (interrupted query)*/
        /* send legal query followed by another command other than a read query response */
        write_IO (":CHANnel2:DISPlay?");
        write_IO (":CHANnel2:DISPlay OFF");

        /* Generate command error - send illegal header */
        write_IO (":CHANnel:DISPlay OFF");

        /* get instrument ID - allow time for SRQ to set */
        write_IO ("*IDN?");
        bytes_read = read_IO (buf,256L);

        /* add NULL to end of string */
        buf [bytes_read] = '\0';

        printf ( "%s\n", buf);

        srq_asserted = check_SRQ ( );

        if ( srq_asserted )
          srq_handler ( );

} /* end create_SRQ ( ) */
```

Listings of the Sample Programs

Listings of the C sample programs in this section include:

- hpibdecl.h
- init.c
- gen_srq.c
- srq.c
- learnstr.c
- sicl_IO.c
- natl_IO.c

Listings of the HP BASIC sample programs in this section include:

- init.bas
- srq.bas
- lrn_str.bas

Read the README File Before Using the Sample Programs

Before using the sample programs, be sure to read the README file on the disk that contains the sample programs.

hpib_decl.h Sample Program

```
/* hpibdecl.h */

/*
*  This file includes necessary prototypes and declarations for
*  the example programs for the Agilent 86100A */
*/

/*
*  User must indicate which HPIB card (HP or National) is being used.
*  Also, if using a National card, indicate which version of windows
*  (WIN31 or WIN95) is being used.
*/


#define HP                              /* Uncomment if using HP interface card */
/* #define NATL */

/* #define WIN31 */                     /* For National card ONLY - select windows version */
#define WIN95

#ifdef HP
#include <sicl.h>
#else
        #ifdef WIN95
        #include <windows.h>            /* include file for Windows 95 */
        #include <decl-32.h>
        #else
        #include <windecl.h>            /* include file for Windows 3.1 */
        #endif
#endif

#define CME 32
#define EXE 16
#define DDE  8
#define QYE  4

#define SRQ_BIT 64
#define MAX_LRNSTR 14000
#define MAX_LENGTH 4096
#define MAX_INT 4192


#ifdef HP
#define DEVICE_ADDR "hpib7,7"
#define INTERFACE "hpib7"
#else
#define INTERFACE "hpib0"

#define board_index 0
#define prim_addr 7
#define second_addr 0
```

```
#define timeout 13
#define eoi_mode  1
#define eos_mode 0
#endif


#define TRUE 1
#define FALSE 0


/* GLOBALS */
#ifdef HP
        INST bus;
        INST scope;
#else
        int bus;
        int scope;
#endif

/* HPIB prototypes */
void init_IO ( );
void write_IO ( void* );
void write_lrnstr ( void*, long );
int read_IO ( void*, unsigned long );
int check_SRQ ( );
unsigned char read_status ( );
void close_IO ( );
void hpiberr ( );


void srq_handler ( );
```

init.c Sample Program

```
/* init. c */

/*
*  Command Order Example.  This program demonstrates the order of commands
*  suggested for operation of the Agilent 86100A analyzer via HPIB.
*  This program initializes the scope, acquires data, performs
*  automatic measurements, and transfers and stores the data on the
*  PC as time/voltage pairs in a comma-separated file format useful
*  for spreadsheet applications. It assumes a SICL INTERFACE exists
*  as 'hpib7' and an Agilent 86100A analyzer at address 7.
*  It also requires the cal signal attached to Channel 1.
*
*  See the README file on the demo disk for development and linking information.
*/

#include <stdio.h>                /* location of: printf ( ) */
#include <stdlib.h>               /* location of: atof(), atoi ( ) */
#include "hpibdecl.h"             /* prototypes, global declarations, constants */

void initialize ( );              /* initialize the scope */
void acquire_data ( );            /* digitize signal */
void auto_measurements ( );       /* perform built-in automatic measurements */
void transfer_data ( );           /* transfers waveform data from scope to PC */
void convert_data ( );            /* converts data to time/voltage values */
void store_csv ( );               /* stores time/voltage pairs to comma-separated variable file format */


/* GLOBALS */
int count;
double xorg,xref,xinc;            /* values necessary for conversion of data */
double yorg,yref,yinc;
int Acquired_length;
char data [MAX_LENGTH];           /* data buffer */
double time_value [MAX_LENGTH];/* time value of data */
double volts [MAX_LENGTH];        /* voltage value of data */

void main( void )
{
        /* initialize interface and device sessions */
        /* note: routine found in sicl_IO.c or natl_IO.c  */
        init_IO ( );


        initialize ( );           /* initialize the scope and interface and set up SRQ  */
        acquire_data ( );         /* capture the data */
        auto_measurements ( );    /* perform automated measurements on acquired data */
        transfer_data ( );        /* transfer waveform data to the PC from scope */
        convert_data ( );         /* convert data to time/voltage pairs */
        store_csv ( );            /* store the time/voltage pairs as csv file */
        close_IO ( );             /* close interface and device sessions */
                                  /* note: routine found in sicl_IO.c or natl_IO.c  */
} /* end main ( ) */
```

```
/*
*  Function name:  initialize
*  Parameters:  none
*  Return value:  none
*  Description:  This routine initializes the analyzer for proper
*  acquisition of data. The instrument is reset to a known state and the
*  interface is cleared. System headers are turned off to allow faster
*  throughput and immediate access to the data values requested by queries.
*  The analyzer time base, channel, and trigger subsystems are then
*  configured. Finally, the acquisition subsystem is initialized.
*/

void initialize (  )
{
        write_IO ("*RST");                      /* reset scope - initialize to known state */
        write_IO ("*CLS");                      /* clear status registers and output queue */

        write_IO (":SYSTem:HEADer OFF"); /* turn off system headers */



        /* initialize time base parameters to center reference, 2 ms full-scale (200 us/div), and 20 us delay */
        write_IO (":TIMebase:REFerence CENTer;RANGe 2e-3;POSition 20e-6");


        /* initialize Channel1 1.6V full-scale (200 mv/div); offset -400mv */
        write_IO (":CHANnel1:RANGe 1.6;OFFSet -400e-3");


        /* initialize trigger info: channel1 signal on positive slope at 300mv */
        write_IO (":TRIGger:SOURce FPANel;SLOPe POSitive");
        write_IO (":TRIGger:LEVel-0.40");



        /* initialize acquisition subsystem */
        /* Real time acquisition - no averaging; record length 4096 */
        write_IO (":ACQuire:AVERage OFF;POINts 4096");


} /* end initialize ( ) */

/*
*  Function name:  acquire_data
*  Parameters: none
*  Return value:  none
*  Description:  This routine acquires data according to the current instrument settings.
*/
void acquire_data ( )
{
/*
*  The root level :DIGitize command is recommended for acquisition of new
*  data. It will initialize data buffers, acquire new data, and ensure that
*  acquisition criteria are met before acquisition of data is stopped.
*  The captured data is then available for measurements, storage, or transfer
*  to a PC.  Note that the display is automatically turned off by the
*  :DIGitize command and must be turned on to view the captured data.
*/

        write_IO (":DIGitize CHANnel1");
        write_IO (":CHANnel1:DISPlay ON");       /* turn on channel 1 display which is turned off by the :DIGitize command */

} /*  end acquire_data() */

/*
*  Function name:  auto_measurements
*  Parameters:  none
*  Return value:  none
*  Description:  This routine performs automatic measurements of volts
*  peak-to-peak and period on the acquired data. It also demonstrates
*  two methods of error detection when using automatic measurements.
*/

void auto_measurements ( )
{
        float period, vpp;
        unsigned char vpp_str[16];
        unsigned char period_str[16];
        int bytes_read;

/*
*  Error checking on automatic measurements can be done using one of two methods.
*  The first method requires that you turn on results in the Measurements
*  subsystem using the command :MEASure:SEND ON.  When this is on, the analyzer
*  will return the measurement and a result indicator.  The result flag is zero
*  if the measurement was successfully completed, otherwise a non-zero value is
*  returned which indicates why the measurement failed. See the Programmer's Manual
*  for descriptions of result indicators.

*  The second method simply requires that you check the return value of the
*  measurement. Any measurement not made successfully will return with the value
*  +9.999E37. This could indicate that either the measurement was unable to be
*  performed, or that insufficient waveform data was available to make the
*  measurement.

* METHOD ONE - turn on results to indicate whether the measurement completed
* successfully. Note that this requires transmission of extra data from the scope.
*/

        write_IO (":MEASure:SEND ON");                  /* turn results on */

        /* query -- volts peak-to-peak channel 1*/
        write_IO (":MEASure:VPP? CHANnel1");

        bytes_read = read_IO (vpp_str,16L);             /* read in value and result flag */

        if (vpp_str[bytes_read-2] != '0')
          printf ("Automated vpp measurement error with result %c\n", vpp_str[bytes_read-2]);
        else
          printf ("VPP is %f\n", (float)atof (vpp_str));
        write_IO (":MEASure:PERiod? CHANnel1");          /* period channel 1 */
        bytes_read = read_IO (period_str,16L);           /* read in value and result flag */

        if (period_str[bytes_read-2] != '0')
          printf ("Automated period measurement error with result %c\n", period_str [bytes_read-2]);
        else
          printf ("Period is %f\n", (float) atof (period_str));


/* METHOD TWO - perform automated measurements and error checking with :MEAS:SEND OFF */

        period = (float) 0;
        vpp = (float) 0;

        /* turn off results */
        write_IO (":MEASure:SEND OFF");

        write_IO (":MEASure:PERiod? CHANnel1");          /* period channel 1 */
        bytes_read = read_IO (period_str,16L);           /* read in value and result flag */

        period = (float) atof (period_str);

        if ( period > 9.99e37 )
          printf ("\nPeriod could not be measured.\n");
        else
          printf ("\nThe period of channel 1 is %f seconds.\n", period );

        write_IO (":MEASure:VPP? CHANnel1");
        bytes_read = read_IO ( vpp_str,16L );

        vpp = (float) atof (vpp_str);

        if ( vpp > 9.99e37 )
          printf ("Peak-to-peak voltage could not be measured.\n");
        else
          printf ("The voltage peak-to-peak is %f volts.\n", vpp );

} /* end auto_measurements ( ) */

/*
*  Function name:  transfer_data
*  Parameters:  none
*  Return value:  none
*  Description:  This routine transfers the waveform conversion factors and waveform data to the PC.
*/

void transfer_data (  )
{
        int header_length;
        char header_str[8];
        char term;

        char xinc_str[32],xorg_str[32],xref_str[32];
        char yinc_str[32],yref_str[32],yorg_str[32];
        int bytes_read;

        /* waveform data source channel 1 */
        write_IO (":WAVeform:SOURce CHANnel1");
        /* setup transfer format */
        write_IO (":WAVeform:FORMat BYTE");
      /* request values to allow interpretation of raw data */
        write_IO (":WAVeform:XINCrement?");
        bytes_read = read_IO (xinc_str,32L);
        xinc = atof (xinc_str);

        write_IO (":WAVeform:XORigin?");
        bytes_read = read_IO (xorg_str,32L);
        xorg = atof (xorg_str);

        write_IO (":WAVeform:XREFerence?");
        bytes_read = read_IO (xref_str,32L);
        xref = atof (xref_str);

        write_IO (":WAVeform:YINCrement?");
        bytes_read = read_IO (yinc_str,32L);
        yinc = atof (yinc_str);

        write_IO (":WAVeform:YORigin?");
        bytes_read = read_IO (yorg_str,32L);
        yorg = atof (yorg_str);

        write_IO (":WAVeform:YREFerence?");
        bytes_read = read_IO (yref_str,32L);
        yref = atof (yref_str);

        write_IO (":WAVeform:DATA?");              /* request waveform data */
        bytes_read = read_IO (data,1L);            /* ignore leading # */
        bytes_read = read_IO (header_str,1L);      /* input byte counter */
        header_length = atoi (header_str);

        /* read number of points - value in bytes */
        bytes_read = read_IO (header_str,(long)header_length);

        Acquired_length = atoi (header_str);       /* number of bytes */

        bytes_read = read_IO (data,Acquired_length);  /* input waveform data */
        bytes_read = read_IO (&term,1L);           /* input termination character */

} /* end transfer_data ( ) */

/*
*  Function name:  convert_data
*  Parameters:  none
*  Return value:  none
*  Description:  This routine converts the waveform data to time/voltage
*  information using the values that describe the waveform.  These values are
*  stored in global arrays for use by other routines.
*/

void convert_data ( )
```

```
{
        int i;

        for (i = 0; i < Acquired_length; i++)
       {
          time_value[i] = ((i - xref) * xinc) + xorg;    /* calculate time info */
          volts[i] = ((data[i] - yref) * yinc) + yorg;  /* calculate volt info */
       }
} /* end convert_data ( ) */

/*
*  Function name:  store_csv
*  Parameters:  none
*  Return value:  none
*  Description:  This routine stores the time and voltage information about
*  the waveform as time/voltage pairs in a comma-separated variable file
*  format.
*/

void store_csv ( )
{
        FILE *fp;
        int i;

        fp = fopen ("pairs.csv","wb");  /* open file in binary mode - clear file if already exists */
        if (fp != NULL)
        {
          for (i = 0; i < Acquired_length; i++)
          {
            /* write time,volt pairs to file */
            fprintf ( fp,"%e,%lf\n",time_value[i],volts[i]);

          }
          fclose ( fp );              /* close file */
        }
        else
          printf ("Unable to open file 'pairs.csv'\n");

}  /* end store_csv ( ) */
```

gen_srq.c Sample Program

```
/* gen_srq.c */

/*
*  This example programs initializes the Agilent 86100A scope, runs an
*  autoscale, then generates and responds to a Service Request from the
*  scope. The program assumes an Agilent 86100A at address 7, an interface card
*  at interface select code 7, and a signal source attached to channel 1.
*/

#include <stdio.h>                    /* location of: printf ( ) */
#include "hpibdecl.h"

void initialize ( );
void setup_SRQ ( );
void create_SRQ ( );

void main ( void )
{
        init_IO ( );                  /* initialize interface and device sessions */
        initialize ( );               /* initialize the scope and interface */
        setup_SRQ ( );                /* enable SRQs on scope and set up SRQ handler */
        create_SRQ ( );               /* generate SRQ */
        close_IO ( );                 /* close interface and device sessions */

} /* end main ( ) */

/*
*  Function name:  initialize
*  Parameters:  none
*  Return value:  none
*  Description:  This routine initializes the analyzer for proper acquisition of data.
* The instrument is reset to a known state and the interface is cleared.
* System headers are turned off to allow faster throughput and immediate access
* to the data values requested by queries. The analyzer performs an autoscale to acquire waveform data.
*/
void initialize (  )
{
        write_IO ("*RST");                  /* reset scope - initialize to known state */
        write_IO ("*CLS");                  /* clear status registers and output queue */
        write_IO (":SYSTem:HEADer OFF"); /* turn off system headers */
        write_IO (":AUToscale");            /* perform autoscale */

} /* end initialize ( ) */

/*
*  Function name:  setup_SRQ
*  Parameters:  none
*  Return value:  none
*  Description:  This routine initializes the device to generate Service
*  Requests. It sets the Service Request Enable Register Event Status Bit
*  and the Standard Event Status Enable Register to allow SRQs on Command
*  or Query errors.
```

```
*/

void setup_SRQ (  )
{
          /* Enable Service Request Enable Register - Event Status Bit */
          write_IO ("*SRE 32");

          /* Enable Standard Event Status Enable Register enable Command Error - bit 4 - value 32 Query Error - bit 1 - value 4 */
          write_IO ("*ESE 36");

} /* end setup_SRQ ( ) */

/*
*  Function name:  create_SRQ
*  Parameters:  none
*  Return value:  none
*  Description:  This routine sends two illegal commands to the scope which will generate an
* SRQ and will place two error strings in the error queue. The scope ID is requested to allow
* time for the SRQ to be generated.  The ID string will contain a leading character which
* is the response placed in the output queue by the interrupted query.
*/
void create_SRQ ( )
{
          char buf [256] = { 0 };  //read buffer for id string
          int bytes_read = 0;
          int srq_asserted;

          /* Generate query error (interrupted query)*/
          /* send legal query followed by another command other than a read query response */
          write_IO (":CHANnel2:DISPlay?");

          write_IO (":CHANnel2:DISPlay OFF");

          /* Generate command error - send illegal header */
          write_IO (":CHANnel:DISPlay OFF");

          /* get instrument ID - allow time for SRQ to set */
          write_IO ("*IDN?");
          bytes_read = read_IO (buf,256L);

          /* add NULL to end of string */
          buf [bytes_read] = '\0';

          printf ( "%s\n", buf);
          srq_asserted = check_SRQ ( );
          if ( srq_asserted )
            srq_handler ( );
} /* end create_SRQ ( ) */
```

srq.c Sample Program

```
/* file:  srq.c */

/*  This file contains the code to handle Service Requests from an HPIB device */

#include <stdio.h>              /* location of printf ( ), fopen ( ), and fclose ( ) */
#include "hpibdecl.h"

/*
*  Function name:  srq_handler
*  Parameters:  none
*  Return value:  none
*  Description:  This routine services the scope when an SRQ is generated.
* An error file is opened to receive error data from the scope.
*/

void srq_handler ( )
     {
        FILE *fp;
        unsigned char statusbyte = 0;
        int i =0;
        int more_errors = 0;
        char error_str[64] ={0};
        int bytes_read;
        int srq_asserted = TRUE;

        srq_asserted = check_SRQ ( );

        while (srq_asserted)
        {
        statusbyte = read_status ( );

        if ( statusbyte & SRQ_BIT )
        {
          fp = fopen ( "error_list","wb" );                    /* open error file */
          if (fp == NULL)
            printf ("Error file could not be opened.\n");
     /* read error queue until no more errors */
          more_errors = TRUE;
          while ( more_errors )
          {
            write_IO (":SYSTEM:ERROR? STRING");
            bytes_read = read_IO (error_str, 64L);

            error_str[bytes_read] = '\0';

            /* write error msg to std IO */
            printf ("Error string:%s\n", error_str );

          if (fp != NULL)
            /* write error msg to file*/
            fprintf (fp,"Error string:%s\n", error_str );
```

```
                if ( error_str[0] == '0' )
                {
                   /* Clear event registers and queues,except output */
                  write_IO("*CLS");

                   more_errors = FALSE;

                      if ( fp != NULL)
                         fclose ( fp );
                }
                for (i=0;i<64;i++)                                  /* clear string */
                   error_str[i] = '\0';

             } /* end while (more_errors) */
          }
          else
          {
             printf (" SRQ not generated by scope.\n ");           /* scope did not cause SRQ */
          }
          srq_asserted = check_SRQ ( );                            /* check for SRQ line status */

          }/* end while ( srq_asserted ) */

}/* end srq_handler */
```

learnstr.c Sample Program

```
/* learnstr.c */

/*
*  This example program initializes the Agilent 86100A scope, runs autoscale to
*  acquire a signal, queries for the learnstring, and stores the learnstring
*  to disk.  It then allows the user to change the setup, then restores the
*  original learnstring. It assumes that a signal is attached to the scope.
*/

#include <stdio.h>                    /* location of: printf ( ), fopen ( ), fclose ( ), fwrite ( ),getchar */
#include "hpibdecl.h"

void initialize ( );
void store_learnstring ( );
void change_setup ( );
void get_learnstring ( );

void main ( void )
{
        init_IO ( );                  /* initialize device and interface  */
                                      /* Note: routine found in sicl_IO.c or natl_IO.c */
        initialize ( );               /* initialize the scope and interface, and set up SRQ */
        store_learnstring ( );        /* request learnstring and store */
        change_setup ( );             /* request user to change setup */
        get_learnstring ( );          /* restore learnstring */
        close_IO ( );                 /* close device and interface sessions */
                                      /* Note: routine found in sicl_IO.c or natl_IO.c */


} /* end main */
/*
*  Function name:  initialize
*  Parameters:  none
*  Return value:  none
*  Description:  This routine initializes the analyzer for proper acquisition of data.
* The instrument is reset to a known state and the interface is cleared.
* System headers are turned off to allow faster throughput and immediate access to the data values requested by queries.
*  Autoscale is performed to acquire a waveform. The signal is then
*  digitized, and the channel display is turned on following the acquisition.
*/

void initialize (  )
{
        write_IO ("*RST");            /* reset scope - initialize to known state */
        write_IO ("*CLS");            /* clear status registers and output queue */

        write_IO (":SYSTem:HEADer ON");/* turn on system headers */

        /* initialize Timebase parameters to center reference, 2 ms full-scale (200 us/div), and 20 us delay */
        write_IO (":TIMebase:REFerence CENTer;RANGe 5e-3;POSition 20e-6");
        /* initialize Channel1 1.6v full-scale (200 mv/div); offset -400mv */
        write_IO (":CHANnel1:RANGe 1.6;OFFSet -400e-3");

        /* initialize trigger info: channel1 signal on positive slope at 300mv */
        write_IO (":TRIGger:SOURce FPANel;SLOPe POSitive");
        write_IO (":TRIGger:LEVel-0.40");

        /* initialize acquisition subsystem */
        /* Real time acquisition - no averaging; record length 4096 */
        write_IO (":ACQuire:AVERage OFF;POINts 4096");

} /* end initialize ( ) */


/*
*  Function name:  store_learnstring
*  Parameters:  none
*  Return value:  none
*  Description:  This routine requests the system setup known as a learnstring.
* The learnstring is read from the scope and stored in a file called Learn2.
*/

void store_learnstring (  )
{
        FILE *fp;
        unsigned char setup[MAX_LRNSTR] ={0};
        int actualcnt = 0;

        write_IO (":SYSTem:SETup?");                    /* request learnstring */
        actualcnt = read_IO (setup, MAX_LRNSTR);

        fp = fopen ( "learn2","wb");


        if ( fp != NULL )
        {
          fwrite ( setup,sizeof (unsigned char), (int) actualcnt,fp);
          printf ("Learn string stored in file Learn2\n");

          fclose ( fp );
        }
        else
          printf ("Error in file open\n");

}/* end store_learnstring */

/*
*  Function name:  change_setup
*  Parameters:  none
*  Return value:  none
*  Description:  This routine places the scope into local mode to allow the customer to change the system setup.
*/

void change_setup ( )
{
        printf ("Please adjust setup and press ENTER to continue.\n");
        getchar();

} /* end change_setup */

/*
*  Function name:  get_learnstring
*  Parameters:  none
*  Return value:  none
*  Description:  This routine retrieves the system setup known as a
*  learnstring from a disk file called Learn2. It then restores the system setup to the scope.
*/

void get_learnstring ( )
{
        FILE *fp;
        unsigned char setup[MAX_LRNSTR];
        unsigned long count = 0;

        fp = fopen ( "learn2","rb");

        if ( fp != NULL )
        {
          count = fread ( setup,sizeof(unsigned char),MAX_LRNSTR,fp);

          fclose ( fp );
        }
        write_lrnstr (setup,count);         /* send learnstring */
        write_IO (":RUN");

}/* end get_learnstring */
```

sicl_IO.c Sample Program

```
/* sicl_IO.c */

#include <stdio.h>                          /* location of: printf ( ) */
#include <string.h>                         /* location of: strlen ( ) */
#include "hpibdecl.h"

/*  This file contains IO and initialization routines for the SICL libraries. */
/*
*  Function name:  init_IO
*  Parameters:  none
*  Return value:  none
*  Description:  This routine initializes the SICL environment. It sets up
*  error handling, opens both an interface and device session, sets timeout
*  values, clears the interface by pulsing IFC, and clears the instrument
*  by performing a Selected Device Clear.
*/

void init_IO ( )
{
        ionerror (I_ERROR_EXIT);            /* set-up interface error handling */

        /* open interface session for verifying SRQ line */
        bus = iopen ( INTERFACE );
        if ( bus == 0 )
          printf ("Bus session invalid\n");

        itimeout ( bus, 20000 );            /* set bus timeout to 20 sec */
        iclear ( bus );                     /* clear the interface - pulse IFC */

        scope = iopen ( DEVICE_ADDR );      /* open the scope device session */
        if ( scope == 0)
          printf ( "Scope session invalid\n");

        itimeout ( scope, 20000 );          /* set device timeout to 20 sec */
        iclear ( scope );                   /* perform Selected Device Clear on scope */

} /* end init_IO */




/*
*  Function name:  write_IO
*  Parameters:  char *buffer which is a pointer to the character string to be
*  output; unsigned long length which is the length of the string to be output
*  Return value:  none
*  Description:  This routine outputs strings to the scope device session
*  using the unformatted I/O SICL commands.
*/
```

```
void write_IO ( void *buffer )
      {
        unsigned long actualcnt;
        unsigned long length;
        int send_end = 1;
        length = strlen ( buffer );
        iwrite ( scope, buffer, length, send_end, &actualcnt );

} /* end write_IO */

/*
*  Function name:  write_lrnstr
*  Parameters:  char *buffer which is a pointer to the character string to be
*  output; long length which is the length of the string to be output
*  Return value:  none
*  Description:  This routine outputs a learnstring to the scope device
*  session using the unformatted I/O SICL commands.
*/

void write_lrnstr ( void *buffer, long length )
{
        unsigned long actualcnt;
        int send_end = 1;

        iwrite ( scope, buffer, (unsigned long) length,
          send_end, &actualcnt );

} /* end write_lrnstr ( ) */



/*
*  Function name:  read_IO
*  Parameters:  char *buffer which is a pointer to the character string to be
*  input; unsigned long length which indicates the max length of the string to be input
*  Return value:  integer which indicates the actual number of bytes read
*  Description:  This routine inputs strings from the scope device session using SICL commands.
*/

int read_IO (void *buffer,unsigned long length)
{
        int reason;
        unsigned long actualcnt;

        iread (scope,buffer,length,&reason,&actualcnt);

        return( (int) actualcnt );
}

/*
*  Function name: check_SRQ
*  Parameters:  none
*  Return value:  integer indicating if bus SRQ line was asserted
*  Description:  This routine checks for the status of SRQ on the bus and returns a value to indicate the status.
*/
```

```
int check_SRQ( )
{
        int srq_asserted;

          /* check for SRQ line status */
          ihpibbusstatus(bus, I_HPIB_BUS_SRQ, &srq_asserted);

        return ( srq_asserted );

} /* end check_SRQ ( ) */




/*
*  Function name:  read_status
*  Parameters:  none
*  Return value:  unsigned char indicating the value of status byte
*  Description:  This routine reads the scope status byte and returns the status.
*/

unsigned char read_status ( )
{
        unsigned char statusbyte;

        /* Always read the status byte from instrument */
        /* NOTE: ireadstb uses serial poll to read status byte - this should clear bit 6 to allow another SRQ. */

          ireadstb ( scope, &statusbyte );
        return ( statusbyte );

} /* end read_status ( ) */

/*
*  Function name:  close_IO
*  Parameters:  none
*  Return value:  none
*  Description:  This routine closes device and interface sessions for the
*  SICL environment and calls the routine _siclcleanup which de-allocates
*  resources used by the SICL environment.
*/

void close_IO ( )
{
        iclose ( scope );   /* close device session */
        iclose ( bus );     /* close interface session */

        _siclcleanup ( );    /* required for 16-bit applications */

} /* end close_SICL ( ) */
```

natl_IO.c Sample Program

```
/* natl_IO.c */

#include <stdio.h>     /* location of: printf ( ) */
#include <string.h>    /* location of: strlen ( ) */
#include "hpibdecl.h"

/*  This file contains IO and initialization routines for the NI488.2 commands. */
/*
*  Function name:  hpiberr
*  Parameters:  char* - string describing error
*  Return value:  none
*  Description:  This routine outputs error descriptions to an error file.
*/

void hpiberr( char *buffer )
{
        printf ("Error string: %s\n",buffer );

} /* end hpiberr ( ) */

/*
*  Function name:  init_IO
*  Parameters:  none
*  Return value:  none
*  Description:  This routine initializes the NI environment. It sets up error
*  handling, opens both an interface and device session, sets timeout values
*  clears the interface by pulsing IFC, and clears the instrument by performing
*  a Selected Device Clear.
*/

void init_IO ( )
{
        bus = ibfind ( INTERFACE );                /* open and initialize HPIB board */
        if ( ibsta & ERR )
          hpiberr ("ibfind error");

        ibconfig ( bus, IbcAUTOPOLL, 0);           /* turn off autopolling */

        ibsic ( bus );                             /* clear interface - pulse IFC */
        if ( ibsta & ERR )
        {
          hpiberr ( "ibsic error" );
        }

        /* open device session */
        scope = ibdev ( board_index, prim_addr, second_addr, timeout,
                  eoi_mode, eos_mode );
        if ( ibsta & ERR )
        {
          hpiberr ( "ibdev error" );
        }
        ibclr ( scope );                        /* clear the device( scope ) */

        if ( ibsta & ERR)
        {
          hpiberr ("ibclr error" );
        }

} /* end init_IO */

/*
*  Function name:  write_IO
*  Parameters:  void *buffer which is a pointer to the character string to be output
*  Return value:  none
*  Description:  This routine outputs strings to the scope device session.
*/
void write_IO ( void *buffer )
{
        long length;

        length = strlen ( buffer );

        ibwrt ( scope, buffer, (long) length );
        if ( ibsta & ERR )
        {
          hpiberr ( "ibwrt error" );
        }

} /* end write_IO() */



/*
*  Function name:  write_lrnstr
*  Parameters:  void *buffer which is a pointer to the character string to
*  be output; length which is the length of the string to be output
*  Return value:  none
*  Description:  This routine outputs a learnstring to the scope device session.
*/
void write_lrnstr ( void *buffer, long length )
{

        ibwrt ( scope, buffer, (long) length );
        if ( ibsta & ERR )
        {
          hpiberr ( "ibwrt error" );
        }

} /* end write_lrnstr ( ) */

/*
*  Function name:  read_IO
*  Parameters:  char *buffer which is a pointer to the character string to be input;
* unsigned long length which indicates the max length of the string to be input
*  Return value:  integer which indicates the actual number of bytes read
*  Description:  This routine inputs strings from the scope device session.
*/
```

```
int read_IO (void *buffer,unsigned long length)
{
  ibrd (scope, buffer, ( long ) length );

  return ( ibcntl );

} /* end read_IO ( ) */





/*
*  Function name:  check_SRQ
*  Parameters:  none
*  Return value:  integer indicating if bus SRQ line was asserted
*  Description:  This routine checks for the status of SRQ on the bus and
*  returns a value to indicate the status.
*/

int check_SRQ ( )
{
          int srq_asserted;
          short control_lines = 0;

          iblines ( bus, &control_lines);

          if ( control_lines & BusSRQ )
            srq_asserted = TRUE;
          else
            srq_asserted = FALSE;

           return ( srq_asserted );

} /* end check_SRQ ( ) */

/*
*  Function name:  read_status
*  Parameters:  none
*  Return value:  unsigned char indicating the value of status byte
*  Description:  This routine reads the scope status byte and returns the status.
*/
unsigned char read_status ( )
{
          unsigned char statusbyte;

          /* Always read the status byte from instrument */

          ibrsp ( scope,  &statusbyte );

          return ( statusbyte );

} /* end read_status ( ) */
```

```
/*
*  Function name:  close_IO
*  Parameters:  none
*  Return value:  none
*  Description:  This routine closes device session.
*/

void close_IO ( )
{
        ibonl ( scope,0 );                      /* close device session */

} /* end close_IO ( ) */
```

init.bas Sample Program

```
10    !file: init
20    !
30    !
40    ! This program demonstrates the order of commands suggested for operation of
50    ! the Agilent 86100A analyzer via HPIB.  This program initializes the scope, acquires
60    ! data, performs automatic measurements, and transfers and stores the data on the
70    ! PC as time/voltage pairs in a comma-separated file format useful for spreadsheet
80    ! applications.  It assumes an interface card at interface select code 7, an
90    ! Agilent 86100A scope at address 7, and the Agilent 86100A cal signal connected to Channel 1.
100   !
110   !
120   !
130   COM /Io/@Scope,@Path,Interface
140   COM /Raw_data/ INTEGER Data(4095)
150   COM /Converted_data/ REAL Time(4095),Volts(4095)
160   COM /Variables/ REAL Xinc,Xref,Xorg,Yinc,Yref,Yorg
170   COM /Variables/ INTEGER Record_length
180   !
190   !
200   CALL Initialize
210   CALL Acquire_data
220   CALL Auto_msmts
230   CALL Transfer_data
240   CALL Convert_data
250   CALL Store_csv
260   CALL Close
270   END
280   !
290   !!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!
300   !
310   !
320   !                    BEGIN SUBPROGRAMS
330   !
340   !!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!
350   !
360   !
370   !     Subprogram name:  Initialize
380   !     Parameters: none
390   !     Return value: none
400   !     Description:  This routine initializes the interface and the scope.   The instrument
410   !      is reset to a known state and the interface is cleared.  System headers
420   !     are turned off to allow faster throughput and immediate access to the
430   !     data values requested by the queries. The analyzer time base,
440 !     channel, and trigger subsystems are then configured. Finally, the
450 !     acquisition subsystem is initialized.
460 !
470 !
480   SUB Initialize
490   COM /Io/@Scope,@Path,Interface
500   COM /Variables/ REAL Xinc,Xref,Xorg,Yinc,Yref,Yorg
510   COM /Variables/ INTEGER Record_length
```

```
520     Interface=7
530     ASSIGN @Scope TO 707
540     RESET Interface
550     CLEAR @Scope
560     OUTPUT @Scope;"*RST"
570     OUTPUT @Scope;"*CLS"
580     OUTPUT @Scope;":SYSTem:HEADer OFF"
590     !Initialize Timebase: center reference, 2 ms full-scale (200 us/div),
              20 us delay
600     OUTPUT @Scope;":TIMebase:REFerence CENTer;RANGe 2e-3;POSition 20e-6"
610       ! Initialize Channel1:  1.6V full-scale (200mv/div), -415mv offset
620     OUTPUT @Scope;":CHANnel1:RANGe 1.6;OFFSet -415e-3"
630       !Initialize Trigger: Edge trigger, channel1 source at -415mv
640     OUTPUT @Scope;":TRIGger:SOURce FPANel;SLOPe POSitive"
650     OUTPUT @Scope;":TRIGger:LEVel-0.415"
660       ! Initialize acquisition subsystem
665       ! Real time acquisition, Averaging off, memory depth 4096
670     OUTPUT @Scope;":ACQuire:AVERage OFF;POINts 4096"
680     Record_length=4096
690   SUBEND
700   !
710   !
720   !!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!
730   !
740   !
750   !    Subprogram name:  Acquire_data
760   !    Parameters: none
770   !    Return value: none
780   !    Description:  This routine acquires data according to the current instrument
790   !                  setting.  It uses the root level :DIGitize command.  This command
800   !                  is recommended for acquisition of new data because it will initialize
810   !                  the data buffers, acquire new data, and ensure that acquisition
820   !                  criteria are met before acquisition of data is stopped.  The captured
830   !                  data is then available for measurements, storage, or transfer to a
840   !                  PC.  Note that the display is automatically turned off by the :DIGitize
850   !                  command and must be turned on to view the captured data.
860   !
870   !
880   SUB Acquire_data
890   COM /Io/@Scope,@Path,Interface
900   OUTPUT @Scope;":DIGitize CHANnel1"
910   OUTPUT @Scope;":CHANnel1:DISPlay ON"
920   SUBEND
930   !
940   !
950   !!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!
960   !
970   !
980   !    Subprogram name:  Auto_msmts
990   !    Parameters: none
1000   !    Return value: none

1010   !    Description:  This routine performs automatic measurements of volts peak-to-peak
1020   !                  and frequency on the acquired data.  It also demonstrates two methods
1030   !                  of error detection when using automatic measurements.
1040   !
```

init.bas Sample Program

```
1050  !
1060  SUB Auto_msmts
1070  COM /Io/@Scope,@Path,Interface
1080  REAL Period,Vpp
1090  DIM Vpp_str$[64]
1100  DIM Period_str$[64]
1110  Bytes_read=0
1120  !
1130  !    Error checking on automatic measurements can be done using one of two methods.
1140  !    The first method requires that you turn on results in the Measurement subsystem
1150  !    using the command ":MEASure:SEND ON".  When this is on, the scope will return the
1160  !    measurement and a result indicator.  The result flag is zero if the measurement
1170  !    was successfully completed, otherwise a non-zero value is returned which indicates
1180  !    why the measurement failed.  See the Programmer's Manual for descriptions of result
1190  !    indicators.  The second method simply requires that you check the return value of
1200  !    the measurement.  Any measurement not made successfully will return with the value
1210  !    +9.999e37.  This could indicate that either the measurement was unable to be
1220  !    performed or that insufficient waveform data was available to make the measurement.
1230  !
1240  !    METHOD ONE
1250  !
1260      OUTPUT @Scope;":MEASure:SEND ON"           !turn on results
1270       OUTPUT @Scope;":MEASure:VPP? CHANnel1"     !Query volts peak-to-peak
1280      ENTER @Scope;Vpp_str$
1290      Bytes_read=LEN(Vpp_str$)                  !Find length of string
1300      CLEAR SCREEN
1310      IF Vpp_str$[Bytes_read;1]="0" THEN         !Check result value
1320        PRINT
1330        PRINT "VPP is ";VAL(Vpp_str$[1,Bytes_read-1])
1340        PRINT
1350      ELSE
1360        PRINT
1370        PRINT "Automated vpp measurement error with result ";Vpp_str$[Bytes_read;1]
1380        PRINT
1390      END IF
1400  !
1410  !
1420      OUTPUT @Scope;":MEASure:PERiod? CHANnel1"  !Query frequency
1430      ENTER @Scope;Period_str$
1440      Bytes_read=LEN(Period_str$)                  !Find string length
1450      IF Period_str$[Bytes_read;1]="0" THEN          !Determine result value
1460        PRINT
1470        PRINT "Period is ";VAL(Period_str$[1,Bytes_read-1])
1480        PRINT
1490      ELSE
1500        PRINT
1510        PRINT "Automated period measurement error with result ";Period_str$[Bytes_read;1]
1520        PRINT
1530      END IF
1540  !
1550  !
1560  !    METHOD TWO
1570  !
1580      OUTPUT @Scope;":MEASure:SEND OFF"          !turn off results
1590      OUTPUT @Scope;":MEASure:VPP? CHANnel1"     !Query volts peak-to-peak
1600      ENTER @Scope;Vpp
```

```
1610       IF Vpp<9.99E+37 THEN
1620         PRINT
1630         PRINT "VPP is ";Vpp
1640         PRINT
1650       ELSE
1660         PRINT
1670         PRINT "Automated vpp measurement error ";Vpp
1680         PRINT
1690       END IF
1700       OUTPUT @Scope;":MEASure:PERiod? CHANnel1"
1710       ENTER @Scope;Period
1720       IF Freq<9.99E+37 THEN
1730         PRINT
1740         PRINT "Period is ";Period
1750         PRINT
1760       ELSE
1770         PRINT
1780         PRINT "Automated period measurement error";Period
1790         PRINT
1800       END IF
1810  SUBEND
1820  !
1830  !
1840  !!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!
1850  !
1860  !
1870  !    Subprogram name:  Transfer_data
1880  !    Parameters: none
1890  !    Return value: none
1900  !    Description:  This routine transfers the waveform data and conversion factors to
1910  !                  to PC.
1920  !
1930  !
1940  SUB Transfer_data
1950  COM /Io/@Scope,@Path,Interface
1960  COM /Raw_data/ INTEGER Data(4095)
1970  COM /Converted_data/ REAL Time(4095),Volts(4095)
1980  COM /Variables/ REAL Xinc,Xref,Xorg,Yinc,Yref,Yorg
1990  COM /Variables/ INTEGER Record_length
2000  !                                define waveform data source and format
2010  OUTPUT @Scope;":WAVeform:SOURce CHANnel1"
2020  OUTPUT @Scope;":WAVeform:FORMat WORD"
2030  !                                request values needed to convert raw data to real
2040  OUTPUT @Scope;":WAVeform:XINCrement?"
2050  ENTER @Scope;Xinc
2060  OUTPUT @Scope;":WAVeform:XORigin?"
2070  ENTER @Scope;Xorg
2080  OUTPUT @Scope;":WAVeform:XREFerence?"
2090  ENTER @Scope;Xref
2100  OUTPUT @Scope;":WAVeform:YINCrement?"
2110  ENTER @Scope;Yinc
2120  OUTPUT @Scope;":WAVeform:YORigin?"
2130  ENTER @Scope;Yorg
2140  OUTPUT @Scope;":WAVeform:YREFerence?"
2150  ENTER @Scope;Yref
2160  !
```

```
2170  !                                   request data
2180  OUTPUT @Scope;":WAVeform:DATA?"
2190  ENTER @Scope USING "#,1A";First_chr$       !ignore leading #
2200  ENTER @Scope USING "#,1D";Header_length    !input number of bytes in header value
2210  ENTER @Scope USING "#,"&VAL$(Header_length)&"D";Record_length     !Record length in bytes
2220  Record_length=Record_length/2             !Record length in words
2230  ENTER @Scope USING "#,W";Data(*)
2240  ENTER @Scope USING "#,A";Term$            !Enter terminating character
2250  !
2260  SUBEND
2270  !
2280  !
2290  !!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!
2300  !
2310  !
2320  !     Subprogram name:  Convert_data
2330  !     Parameters: none
2340  !     Return value: none
2350  !     Description:  This routine converts the waveform data to time/voltage information
2360  !                 using the values Xinc, Xref, Xorg, Yinc, Yref, and Yorg used to describe
2370  !                 the raw waveform data.
2380  !
2390  !
2400  SUB Convert_data
2410  COM /Io/@Scope,@Path,Interface
2420  COM /Raw_data/ INTEGER Data(4095)
2430  COM /Converted_data/ REAL Time(4095),Volts(4095)
2440  COM /Variables/ REAL Xinc,Xref,Xorg,Yinc,Yref,Yorg
2450  COM /Variables/ INTEGER Record_length
2460  !
2470  FOR I=0 TO Record_length-1
2480       Time(I)=(((I)-Xref)*Xinc)+Xorg
2490       Volts(I)=((Data(I)-Yref)*Yinc)+Yorg
2500  NEXT I
2510  SUBEND
2520  !
2530  !
2540  !!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!
2550  !
2560  !
2570  !     Subprogram name: Store_csv
2580  !     Parameters: none
2590  !     Return value: none
2600  !     Description:  This routine stores the time and voltage information about the waveform
2610  !        as time/voltage pairs in a comma-separated variable file format.
2620  !
2630  !
2640  SUB Store_csv
2650  COM /Io/@Scope,@Path,Interface
2660  COM /Converted_data/ REAL Time(4095),Volts(4095)
2670  COM /Variables/ REAL Xinc,Xref,Xorg,Yinc,Yref,Yorg
2680  COM /Variables/ INTEGER Record_length
2690        !Create a file to store pairs in
2700  ON ERROR GOTO Cont
2710  PURGE "Pairs.csv"
2720 Cont: OFF ERROR
```

```
2730  CREATE "Pairs.csv",Max_length
2740  ASSIGN @Path TO "Pairs.csv";FORMAT ON
2750                            !Output data to file
2760  FOR I=0 TO Record_length-1
2770     OUTPUT @Path;Time(I),Volts(I)
2780  NEXT I
2790  SUBEND
2800  !
2810  !
2820  !!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!
2830  !
2840  !
2850  !     Subprogram name: Close
2860  !     Parameters: none
2870  !     Return value: none
2880  !     Description: This routine closes the IO paths.
2890  !
2900  !
2910  SUB Close
2920  COM /Io/@Scope,@Path,Interface
2930
2940  RESET Interface
2950  ASSIGN @Path TO *
2960  SUBEND
```

srq.bas Sample Program

```
10   !File: srq.bas
20   !
30   !   This program demonstrates how to set up and check Service Requests from
40   !   the scope.  It assumes an interface select code of 7 with a scope at
50   !   address 7.  It also assumes a signal is connected to the scope.
60   !
70   !
80   COM /Io/@Scope,Interface
90   COM /Variables/Temp
100  CALL Initialize
110  CALL Setup_srq
120      ON INTR Interface CALL Srq_handler    !Set up routine to handle interrupt
130      ENABLE INTR Interface;2               !Enable SRQ Interrupt for Interface
140  CALL Create_srq
150  CALL Close
160  END
170  !
180  !!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!
190  !
200  !                   BEGIN SUBPROGRAMS
210  !
220  !!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!
230  !
240  !
250  !     Subprogram name:  Initialize
260  !     Parameters: none
270  !     Return value: none
280  !     Description:   This routine initializes the interface and the scope.
290  !             The instrument is reset to a known state and the interface is
300  !                   cleared.  System headers are turned off to allow faster throughput
310  !                 and immediate access to the data values requested by the queries.
320  !
330  !
340  SUB Initialize
350  COM /Io/@Scope,Interface
360      ASSIGN @Scope TO 707
370      Interface=7
380      RESET Interface
390      CLEAR @Scope
400      OUTPUT @Scope;"*RST"
410      OUTPUT @Scope;"*CLS"
420      OUTPUT @Scope;":SYSTem:HEADer OFF"
430      OUTPUT @Scope;":AUToscale"
440  SUBEND
450  !
460  !
470  !
480  !!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!
490  !
500  !     Subprogram name: Setup_srq
510  !     Parameters: none
```

```
520  !     Return value: none
530    !     Description:  This routine sets up the scope to generate Service Requests.
540  !                  It sets the Service Request Enable Register Event Status Bit
550  !                 and the Standard Event Status Enable REgister to allow SRQs on
560  !              Command or Query errors.
570  !
580  !
590  SUB Setup_srq
600  COM /Io/@Scope,Interface
610      OUTPUT @Scope;"*SRE 32"   !Enable Service Request Enable Registers - Event Status bit
620  !
630  !    Enable Standard Event Status Enable Register:
640  !       enable  bit 4 - Command Error -  value 32
650  !              bit 1 - Query Error - value 4
660      OUTPUT @Scope;"*ESE 36"
670  SUBEND
680  !
690  !
700  !
710  !!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!
720  !
730  !
740  !     Subprogram name:  Create_srq
750  !     Parameters: none
760  !     Return value: none
770  !     Description:  This routine will send an illegal command to the scope to
780  !                 show how to detect and handle an SRQ.  A query is sent to
790  !                 the scope which is then followed by another command causing
800  !                 a query interrupt error.  An illegal command header is then
810  !                 sent to demonstrate how to handle multiple errors in the error queue.
820  !
830  !
840  !
850  SUB Create_srq
860  COM /Io/@Scope,Interface
870      DIM Buf$[256]
880      OUTPUT @Scope;":CHANnel2:DISPlay?"
890      OUTPUT @Scope;":CHANnel2:DISPlay OFF"     !send query interrupt
900      OUTPUT @Scope;":CHANnel:DISPlay OFF"      !send illegal header
910                          ! Do some stuff to allow time for SRQ to be recognized
920                          !
930      OUTPUT @Scope;"*IDN?"   !Request IDN to verify communication
940        ENTER @Scope;Buf$       !NOTE: There is a leading zero to this query response
950      PRINT                  !which represents the response to the interrupted query above
960      PRINT Buf$
970      PRINT
980  SUBEND
990  !
1000  !
1010  !
1020  !!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!
1030  !
1040  !
1050  !     Subprogram name:  Srq_handler
1060  !     Parameters: none
1070  !     Return value: none
```

```
1080  !     Description:  This routine verifies the status of the SRQ line.  It then checks
1090  !                   the status byte of the scope to determine if the scope caused the
1100  !                   SRQ.  Note that using a SPOLL to read the status byte of the scope
1110  !                   clears the SRQ and allows another to be generated.  The error queue
1120  !                   is read until all errors have been cleared.  All event registers and
1130  !                   queues, except the output queue, are cleared before control is returned
1140  !                   to the main program.
1150  !
1160  !
1170  !
1180  SUB Srq_handler
1190       COM /Io/@Scope,Interface
1200       DIM Error_str$[64]
1210       INTEGER Srq_asserted,More_errors
1220     Status_byte=SPOLL(@Scope)
1230     IF BIT(Status_byte,6) THEN
1240            More_errors=1
1250            WHILE More_errors
1260                OUTPUT @Scope;":SYSTem:ERROR? STRING"
1270                ENTER @Scope;Error_str$
1280                PRINT
1290                PRINT Error_str$
1300                IF Error_str$[1,1]="0" THEN
1310                     OUTPUT @Scope;"*CLS"
1320                     More_errors=0
1330                END IF
1340            END WHILE
1350       ELSE
1360           PRINT
1370           PRINT "Scope did not cause SRQ"
1380           PRINT
1390       END IF
1400       ENABLE INTR Interface;2      !re-enable SRQ
1410  SUBEND
1420  !
1430  !
1440  !!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!
1450  !
1460  !     Subprogram name:  Close
1470  !     Parameters: none
1480  !     Return value: none
1490  !     Description:  This routine resets the interface.
1500  !
1510  !
1520  !
1530  SUB Close
1540  COM /Io/@Scope,Interface
1550
1560  RESET Interface
1570  SUBEND
1580  !
1590  !
1600  !!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!
```

lrn_str.bas Sample Program

```
10   !FILE: lrn_str.bas
20   !
30   !THIS PROGRAM WILL INITIALIZE THE SCOPE, AUTOSCALE, AND DIGITIZE THE WAVEFORM
40   !INFORMATION.  IT WILL THEN QUERY THE INSTRUMENT FOR THE LEARNSTRING AND WILL
50   !SAVE THE INFORMATION TO A FILE.  THE PROGRAM WILL THEN PROMPT YOU TO CHANGE
60   !THE SETUP THEN RESTORE THE ORIGINAL LEARNSTRING CONFIGURATION. IT ASSUMES
70   !AN Agilent 86100A at ADDRESS 7, HPIB INTERFACE at 7, AND THE CAL SIGNAL ATTACHED TO
80   !CHANNEL 1.
90   !
100  !
110  COM /Io/@Scope,@Path,Interface
120  COM /Variables/Max_length
130  CALL Initialize
140  CALL Store_lrnstr
150  CALL Change_setup
160  CALL Get_lrnstr
170  CALL Close
180  END
190  !
200  !
210  !!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!
220  !
230  !                 BEGIN SUBROUTINES
240  !
250  !!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!
260  !    Subprogram name:  Initialize
270  !    Parameters: none
280  !    Return value: none
290  !    Description:  This routine initializes the path descriptions and resets the
300  !                interface and the scope.  It performs an autoscale on the signal,
310  !                acquires the data on channel 1, and turns on the display.
320  !                NOTE:  This routine also turns on system headers.  This allows the
330  !                string ":SYSTEM:SETUP " to be returned with the learnstring so the
340  !                return string is in the proper format.
350  !
360  SUB Initialize
370      COM /Io/@Scope,@Path,Interface
380      COM /Variables/Max_length
390      Max_length=14000
400      ASSIGN @Scope TO 707
410      Interface=7
420      RESET Interface
430      CLEAR @Scope
440      OUTPUT @Scope;"*RST"
450      OUTPUT @Scope;"*CLS"
460      OUTPUT @Scope;":SYSTem:HEADer ON"
470      OUTPUT @Scope;":AUToscale"
480  SUBEND
490  !
500  !
510  !!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!
```

```
520  !
530  !
540  !     Subprogram name: Store_lrnstr
550  !     Parameters: none
560  !     Return value: none
570  !     Description:  This routine creates a file in which to store the learnstring
580  !                  configuration (Filename:Lrn_strg).  It requests the learnstring
590  !                 and inputs the configuration to the PC.  Finally, it stores the
600  !              configuration to the file.
610  !
620  SUB Store_lrnstr
630       COM /Io/@Scope,@Path,Interface
640       COM /Variables/Max_length
650       ON ERROR GOTO Cont
660       PURGE "Lrn_strg"
670 Cont:  OFF ERROR
680       CREATE BDAT "Lrn_strg",1,14000
690       DIM Setup$[14000]
700       ASSIGN @Path TO "Lrn_strg"
710       OUTPUT @Scope;":SYSTem:SETup?"
720       ENTER @Scope USING "-K";Setup$
730       OUTPUT @Path,1;Setup$
740       CLEAR SCREEN
750       PRINT "Learn string stored in file: Lrn_strg"
760  SUBEND
770  !
780  !
790  !!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!
800  !
810  !     Subprogram name: Change_setup
820  !     Parameters: none
830  !     Return value: none
840  !     Description:  This subprogram requests that the user change the
850  !                  scope setup, then press a key to continue.
860  !
870  !
880  SUB Change_setup
890       COM /Io/@Scope,@Path,Interface
900
910       PRINT
920       PRINT "Please adjust setup and press Continue to resume."
930       PAUSE
940  SUBEND
950  !
960  !
970  !!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!
980  !
990  !     Subprogram name: Get_lrnstr
1000  !     Parameters: none
1010  !     Return value: none
1020  !     Description:  This subprogram loads a learnstring from the
1030  !                  file "Lrn_strg" to the scope.
1040  !
1050  !
1060  SUB Get_lrnstr
1070       COM /Io/@Scope,@Path,Interface
```

```
1080      COM /Variables/Max_length
1090      DIM Setup$[14000]
1100      ENTER @Path,1;Setup$
1110      OUTPUT @Scope USING "#,-K";Setup$
1120      OUTPUT @Scope;":RUN"
1130  SUBEND
1140  !
1150  !
1160  !!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!
1170  !
1180  !
1190  !     Subprogram name: Close
1200  !     Parameters: none
1210  !     Return value: none
1220  !     Description:  This routine resets the interface, and closes all I/O paths.
1230  !
1240  !
1250  !
1260  SUB Close
1270  COM /Io/@Scope,@Path,Interface
1280
1290  RESET Interface
1300  ASSIGN @Path TO *
1310  SUBEND
1320  !
1330  !!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!
```

7

Common Commands

Common Commands

Common commands are defined by the IEEE 488.2 standard. They control generic device functions that are common to many different types of instruments. Common commands can be received and processed by the analyzer, whether they are sent over the GPIB as separate program messages or within other program messages.

Receiving Common Commands

Common commands can be received and processed by the analyzer, whether they are sent over the GPIB as separate program messages or within other program messages. If a subsystem is currently selected and a common command is received by the analyzer, the analyzer remains in the selected subsystem. For example, if the program message

```
"ACQUIRE:AVERAGE ON;*CLS;COUNT 1024"
```

is received by the analyzer, the analyzer enables averaging, clears the status information, then sets the number of averages without leaving the selected subsystem.

Status Registers

The following two status registers used by common commands have an enable (mask) register. By setting bits in the enable register, the status information can be selected for use. Refer to Chapter 4, “Status Reporting” for a complete discussion of status.

Table 7-1. Status Registers

Status Register	Enable Register
Event Status Register	Event Status Enable Register
Status Byte Register	Service Request Enable Register

Common Commands

*CLS (Clear Status)

Command *CLS

The *CLS command clears all status and error registers.

Example This example clears the status data structures of the analyzer.

```
10 OUTPUT 707;"*CLS"
20 END
```

See Also Refer to Chapter 4, “Status Reporting” for a complete discussion of status.

*ESE (Event Status Enable)

Command *ESE <mask>

The *ESE command sets the Standard Event Status Enable Register bits.

<mask> An integer, 0 to 255, representing a mask value for the bits to be enabled in the Standard Event Status Register as shown in Table 7-2 on page 7-5.

Example This example enables the User Request (URQ) bit of the Standard Event Status Enable Register. When this bit is enabled and a front-panel key is pressed, the Event Summary bit (ESB) in the Status Byte Register is also set.

```
10 OUTPUT 707;"*ESE 64"
20 END
```

Query *ESE?

The *ESE? query returns the current contents of the Standard Event Status Enable Register.

Returned Format <mask><NL>

<mask> An integer, +0 to +255 (the plus sign is also returned), representing a mask value for the bits enabled in the Standard Event Status Register as shown in Table 7-2 on page 7-5.

Example

This example places the current contents of the Standard Event Status Enable Register in the numeric variable, Event. The value of the variable is printed on the computer's screen.

```
10 OUTPUT 707;"*ESE?"
20 ENTER 707;Event
30 PRINT Event
40 END
```

The Standard Event Status Enable Register contains a mask value for the bits to be enabled in the Standard Event Status Register. A "1" in the Standard Event Status Enable Register enables the corresponding bit in the Standard Event Status Register. A "0" in the enable register disables the corresponding bit.

Table 7-2. Standard Event Status Enable Register Bits

Bit	Weight	Enables	Definition
7	128	PON - Power On	Indicates power is turned on.
6	64	URQ - User Request	Not used. Permanently set to zero.
5	32	CME - Command Error	Indicates whether the parser detected an error.
4	16	EXE - Execution Error	Indicates whether a parameter was out-of-range, or was inconsistent with the current settings.
3	8	DDE - Device Dependent Error	Indicates whether the device was unable to complete an operation for device-dependent reasons.
2	4	QYE - Query Error	Indicates if the protocol for queries has been violated.
1	2	RQC - Request Control	Indicates whether the device is requesting control.
0	1	OPC - Operation Complete	Indicates whether the device has completed all pending operations.

See Also

Refer to Chapter 4, “Status Reporting” for a complete discussion of status.

*ESR? (Event Status Register)

Query

*ESR?

The *ESR? query returns the contents of the Standard Event Status Register. Reading this register clears the Standard Event Status Register, as does *CLS.

Returned Format

<status><NL>

<status>

An integer, 0 to 255, representing the total bit weights of all bits that are high at the time you read the register.

Example

This example places the current contents of the Standard Event Status Register in the numeric variable, Event, then prints the value of the variable to the computer's screen.

```
10 OUTPUT 707;"*ESR?"
20 ENTER 707;Event
30 PRINT Event
40 END
```

Table 7-3 lists each bit in the Event Status Register and the corresponding bit weights.

Table 7-3. Standard Event Status Register Bits

Bit	Bit Weight	Bit Name	Condition
7	128	PON	1 = OFF to ON transition has occurred.
6	64		Not Used. Permanently set to zero.
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	RQC	0 = request control - 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)

Query

*IDN?

The *IDN? query returns the company name, analyzer model number, serial number, and software version by returning the following string:

AGILENT TECHNOLOGIES,86100A,<USXXXXXXXX>,<Rev #>

<USXXXXXXXX>

Specifies the serial number of the analyzer. The first four digits and letter are the serial prefix, which is the same for all identical analyzers. The last five digits are the serial suffix, which is assigned sequentially, and is different for each analyzer.

<Rev #>

Specifies the software version of the analyzer, and is the revision number.

Returned Format

AGILENT TECHNOLOGIES,86100A,USXXXXXXXX,A.XX.XX<NL>

Example

This example places the analyzer's identification information in the string variable, Identify$, then prints the identification information to the computer screen.

```
10 DIM Identify$[50]          !Dimension variable
20 OUTPUT 707;"*IDN?"
30 ENTER 707;Identify$
40 PRINT Identify$
50 END
```

*LRN? (Learn)

Query

*LRN?

The *LRN? query returns a string that contains the analyzer's current setup. The analyzer's setup can be stored and sent back to the analyzer at a later time. This setup string should be sent to the analyzer just as it is. It works because of its embedded ":SYStem:SETup" header.

Returned Format

:SYSTem:SETup <setup><NL>

<setup>

This is a definite length arbitrary block response specifying the current analyzer setup. The block size is subject to change with different firmware revisions.

Example

This example sets the scope's address and asks for the learn string, then determines the string length according to the IEEE 488.2 block specification. It then reads the string and the last EOF character.

```
10 ! Set up the scope's address and
20 ! ask for the learn string...
30 ASSIGN @Scope TO 707
40 OUTPUT @Scope:"*LRN?"
50 !
60 ! Search for the # sign.
70 !
80 Find_pound_sign: !
90 ENTER @Scope USING "#,A";Thischar$
100 IF Thischar$<>"#" THEN Find_pound_sign
110 !
120 ! Determine the string length according
130 ! to the IEEE 488.2 # block spec.
140 ! Read the string then the last EOF char.
150 !
160 ENTER @Scope USING "#,D";Digit_count
170 ENTER @Scope USING "#,"&VAL$(Digit_count)&"D";Stringlength
180 ALLOCATE Learn_string$[Stringlength+1]
190 ENTER @Scope USING "-K";Learn_string$
200 OUTPUT 707;":syst:err?"
210 ENTER 707;Errornum
220 PRINT "Error Status=";Errornum
```

***LRN? Returns Prefix to Setup Block**

The *LRN query always returns :SYSTem:SETup as a prefix to the setup block. The SYSTem:HEADer command has no effect on this response.

See Also

SYSTem:SETup command and query. When HEADers and LONGform are ON, the SYSTem:SETup command performs the same function as the *LRN query. Otherwise, *LRN and SETup are not interchangeable.

*OPC (Operation Complete)

Command

```
*OPC
```

The *OPC command sets the operation complete bit in the Standard Event Status Register when all pending device operations have finished.

Example

This example sets the operation complete bit in the Standard Event Status Register when the PRINT operation is complete.

```
10 OUTPUT 707;":PRINT;*OPC"
20 END
```

Query

```
*OPC?
```

The *OPC? query places an ASCII character “1” in the analyzer's output queue when all pending selected device operations have finished.

Returned Format

```
1<NL>
```

Example

This example places an ASCII character “1” in the analyzer's output queue when the SINGle operation is complete. Then the value in the output queue is placed in the numeric variable “Complete.”

```
10 OUTPUT 707;":SINGle;*OPC?"
20 ENTER 707;Complete
30 PRINT Complete
40 END
```

The *OPC query allows synchronization between the computer and the analyzer by using the message available (MAV) bit in the Status Byte, or by reading the output queue. Unlike the *OPC command, the *OPC query does not affect the OPC Event bit in the Standard Event Status Register.

*OPT? (Option)

Query *OPT?

The OPT? query returns a string with a list of installed options. If no options are installed, the string will have a 0 as the first character.

The length of the returned string may increase as options become available in the future. Once implemented, an option name will be appended to the end of the returned string, delimited by a comma.

Example This example places all options into the string variable, Options$, then prints the option model and serial numbers to the computer's screen.

```
10 DIM Options$[100]
20 OUTPUT 707;"*OPT?"
30 ENTER 707;Options$
40 PRINT Options$
50 END
```

*RCL (Recall)

Command *RCL <register>

The *RCL command restores the state of the analyzer to a setup previously stored in the specified save/recall register. An analyzer setup must have been stored previously in the specified register. Registers 0 through 9 are general-purpose registers and can be used by the *RCL command.

<register> An integer, 0 through 9, specifying the save/recall register that contains the analyzer setup you want to recall.

Example This example restores the analyzer to the analyzer setup stored in register 3.

```
10 OUTPUT 707;"*RCL 3"
20 END
```

See Also SAVe. An error message appears on the analyzer display if nothing has been previously saved in the specified register.

*RST (Reset)

Command *RST

The *RST command places the analyzer in a known state. Table 7-4 lists the reset conditions as they relate to the analyzer commands. This is the same as using the front-panel default setup button.

Example This example resets the analyzer to a known state.

```
10 OUTPUT 707;"*RST"
20 END
```

This following table shows the analyzer's default setup.

Table 7-4. Default Setup (1 of 5)

Acquisition	
Run/Stop	100 ms
	Grid on
	30
	Enabled
	8 hours
	Default legend
	Off
	Off (until the first marker is placed on the screen)
	User selectable if more than one source is available.
	28 ns
	0V
Points/Waveform (Record length)	Automatic - 1350 points
Averaging	Off
# of Averages	16
Trigger	
Source	Front Panel
Bandwidth	2.5 GHz
Hysteresis	Normal
Slope	Positive
Gated Trigger	Off

Table 7-4. Default Setup (2 of 5)

Level	0 V	
Time Base		
Units	Time	
Scale	1 ns/div	
Position	24 ns	
Reference	Left	
Display		
Persistence	Variable (oscilloscope mode)	
	Gray Scale (Infinite) (Eye/Mask mode)	
Persistence Time	100 ms	
Graticule	Grid on	
Intensity	30	
Backlight Saver	Enabled	
Turn off backlight after	8 hours	
Colors	Default legend	
Labels	Off	
Markers		
Mode		
Readout	Off (until the first marker is placed on the screen)	
X1, Y1 source	User selectable if more than one source is available	
X1 position	28 ns	
Y1 position	0V	
X2, Y2 source	User selectable if more than one source is available	
X2 position	24 ns	
Y2 position	0V	
Measure	**Oscilloscope mode**	**Eye/Mask mode**
QuickMeas, Meas.1	V p-p	Extinction ratio
QuickMeas, Meas. 2	Period	Jitter
QuickMeas, Meas. 3	Frequency	Average power
QuickMeas, Meas. 4	Rise time	Crossing %
Start mask test	—	Off
Define Measure		
Thresholds - percent	10%, 50%, 90%	

Table 7-4. Default Setup (3 of 5)

Thresholds - volts	0.0, 1.6, 5.0
Top-Base Definition	Standard
Statistics	Off
Top-Base volts	0.0, 5.0
Measurements	Off
Start Edge	Rising, 1 level, middle
Stop Edge	Falling, 1 level, middle
Eye Window 1	40%
Eye Window 2	60%
Duty cycle distortion format	Time
Extinction ratio format	Decibel
Eye width	Time
Jitter	RMS
Average power	Watts
Waveform	
Memory display	Off
Waveform source	First available channel or memory 1
Memory type	Waveform
Math	
Function	Function 1
Function state	Off
Operator	Magnify
Operand 1	First available channel or memory 1
Operand 2	First available channel or memory 1
Horizontal scaling	Track source
Vertical scaling	Track source
Channel	
Display	On (lowest number installed channel; others are off)
Scale	50 μW/div or 10 mV/div
Offset	0.0 V or 0 W
Units	Volts (or watts)
Filter	Dependent on module
Wavelength	Wavelength 1
Bandwidth	Dependent on module
Histogram	

Table 7-4. Default Setup (4 of 5)

Mode	Off
Axis	Horizontal
Window source	First available channel
Size	Horizontal - 4.0 divisions
	Vertical - 5.0 divisions
X1 position	25 ns
Y1 position	1 division up from bottom, value depends on module
X2 position	33 ns
Y2 position	1 division down from top, value depends on module
Utilities	
Cal Output	5.0 mv
Calibration Details	Off
Self Test	Scope Self Tests
Service Extensions	Off
Remote Interface	Unchanged
Dialog Preferences	Opaque Dialogs
Allow Multiple Active Dialogs	Off
Sound	enabled, volume 48
Limit Test	
Test	Off
Measurement	None
Fail when	Outside
Upper limit	10
Lower limit	-10
Run until	Forever
Run until failures	1 failure
Run until waveforms	1,000,000 waveforms
Store summary	Off
Store screen	Off
Store waveforms	Off
Mask Test	
Test	Off
Scale source	Displayed channel
X1 position	2 divisions from left, 26 ns
1 level	2 divisions down

Table 7-4. Default Setup (5 of 5)

0 level	2 divisions up
Mask margins	Off
Run until	Forever
Failed waveforms	1 failure
Failed samples	1 sample
Waveforms	1,000,000
Samples	1,000,000
Store waveforms	Off
Store summary	Off
Store screen	Off

*SAV (Save)

Command *SAV <register>

The *SAV command stores the current state of the analyzer in a save register.

<register> An integer, 0 through 9, specifying which register to save the current analyzer setup.

Example This example stores the current analyzer setup to register 3.

```
10 OUTPUT 707;"*SAV 3"
20 END
```

See Also *RCL (Recall)

*SRE (Service Request Enable)

Command *SRE <mask>

The *SRE command sets the Service Request Enable Register bits. By setting the *SRE, when the event happens, you have enabled the analyzer's interrupt capability. The scope will then do an SRQ (service request), which is an interrupt.

<mask> An integer, 0 to 255, representing a mask value for the bits to be enabled in the Service Request Enable Register as shown in Table 7-5 on page 7-17.

Example 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 is high.

```
10 OUTPUT 707;"*SRE 16"
20 END
```

Query *SRE?

The *SRE? query returns the current contents of the Service Request Enable Register.

Returned Format <mask><NL>

<mask> An integer, 0 to 255, representing a mask value for the bits enabled in the Service Request Enable Register.

Example

This example places the current contents of the Service Request Enable Register in the numeric variable, Value, then prints the value of the variable to the computer's screen.

```
10 OUTPUT 707;"*SRE?"
20 ENTER 707;Value
30 PRINT Value
40 END
```

The Service Request Enable Register contains a mask value for the bits to be enabled in the Status Byte Register. A “1” in the Service Request Enable Register enables the corresponding bit in the Status Byte Register. A “0” disables the bit.

Table 7-5. Service Request Enable Register Bits

Bit	Weight	Enables
7	128	OPER - Operation Status Register
6	64	Not Used
5	32	ESB - Event Status Bit
4	16	MAV - Message Available
3	8	Not Used
2	4	MSG - Message
1	2	USR - User Event Register
0	1	TRG - Trigger

*STB? (Status Byte)

Query

*STB?

The *STB? query returns the current contents of the Status Byte, including the Master Summary Status (MSS) bit. See Table 7-6 on page 7-18 for Status Byte Register bit definitions.

Returned Format

<value><NL>

<value>

An integer, from 0 to 255.

Example

This example reads the contents of the Status Byte into the numeric variable, Value, then prints the value of the variable to the computer's screen.

```
10 OUTPUT 707;"*STB?"
20 ENTER 707;Value
30 PRINT Value
40 END
```

In response to a serial poll (SPOLL), Request Service (RQS) is reported on bit 6 of the status byte. Otherwise, the Master Summary Status bit (MSS) is reported on bit 6. MSS is the inclusive OR of the bitwise combination, excluding bit 6, of the Status Byte Register and the Service Request Enable Register. The MSS message indicates that the scope is requesting service (SRQ).

Table 7-6. Status Byte Register Bits

Bit	Bit Weight	Bit Name	Condition
7	128	OPER	0 = no enabled operation status conditions have occurred 1 = an enabled operation status condition has occurred
6	64	RQS/MSS	0 = analyzer has no reason for service 1 = analyzer is requesting service
5	32	ESB	0 = no event status conditions have occurred 1 = an enabled event status condition occurred
4	16	MAV	0 = no output messages are ready 1 = an output message is ready
3	8	—	0 = not used
2	4	MSG	0 = no message has been displayed 1 = message has been displayed
1	2	USR	0 = no enabled user event conditions have occurred 1 = an enabled user event condition has occurred
0	1	TRG	0 = no trigger has occurred 1 = a trigger occurred
	0 = False = Low		1 = True = High

*TRG (Trigger)

Command

```
*TRG
```

The *TRG command has the same effect as the Group Execute Trigger message (GET) or RUN command. It acquires data for the active waveform display, if the trigger conditions are met, according to the current settings.

Example

This example starts the data acquisition for the active waveform display according to the current settings.

```
10 OUTPUT 707;"*TRG"
20 END
```

*TST? (Test)

Query

```
*TST?
```

The *TST? query causes the analyzer to perform a self-test, and places a response in the output queue indicating whether or not the self-test completed without any detected errors. Use the :SYSTem:ERRor command to check for errors. A zero indicates that the test passed and a non-zero indicates the self-test failed.

Disconnect Inputs First

You must disconnect all front-panel inputs before sending the *TST? query.

Returned Format

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

<result>

0 for pass; non-zero for fail.

Example

This example performs a self-test on the analyzer and places the results in the numeric variable, Results. The program then prints the results to the computer's screen.

```
10 OUTPUT 707;"*TST?"
20 ENTER 707;Results
30 PRINT Results
40 END
```

If a test fails, refer to the troubleshooting section of the service guide.

The Self-Test takes approximately 3 minutes to complete. When using time-outs in your program, 200 seconds duration is recommended.

*WAI (Wait-to-Continue)

Command *WAI

The *WAI command prevents the analyzer from executing any further commands or queries until all currently executing commands are completed. See *OPC for alternate methods for synchronization.

Example This example executes a single acquisition, and causes the instrument to wait until acquisition is complete before executing any additional commands.

```
10 OUTPUT 707;"SINGle;*WAI"
20 END
```

8

Root Level Commands

Root Level Commands

Root level commands control many of the basic operations of the analyzer that can be selected by pressing the labeled keys on the front panel. These commands are always recognized by the parser if they are prefixed with a colon, regardless of the current tree position. After executing a root level command, the parser is positioned at the root of the command tree.

Status Reporting Data Structures

For any of the Standard Event Status Register bits to generate a summary bit, the bits must be enabled. These bits are enabled by using the *ESE common command to set the corresponding bit in the Standard Event Status Enable Register. URQ in the Event Status Register always returns 0.

To generate a service request (SRQ) interrupt to an external computer, at least one bit in the Status Byte Register must be enabled. These bits are enabled by using the *SRE common command to set the corresponding bit in the Service Request Enable Register. These enabled bits can then set RQS and MSS (bit 6) in the Status Byte Register. In the SRE query, bit 6 always returns 0.

Various root level commands documented in this chapter query and set various registers within the register set.

Root Level Commands

AEEN (Acquisition Limits Event Enable register)

Command :AEEN <mask>

This command sets a mask into the Acquisition Limits Event Enable register. A “1” in a bit position enables the corresponding bit in the Acquisition Limits Event Register to set bit 9 in the Operation Status Register.

<mask> The decimal weight of the enabled bits. Only bit 0 is used at this time; so to enable the COMP bit, set bit 0 to “1” with the AEEN command. Otherwise, set bit 0 to “0”.

Query :AEEN?

The query returns the current decimal value in the Acquisition Limits Event Enable register.

Returned Format [:AEEN] <mask><NL>

ALER? (Acquisition Limits Event Register)

Query :ALER?

This query returns the current value of the Acquisition Limits Event Register as a decimal number and also clears this register.

Bit 0 (COMP) of the Acquisition Limits Event Register is set when the acquisition completes. The acquisition completion criteria are set by the :ACQuire:RUNTil command.

Returned Format [:ALER] <value><NL>

AUToscale

Command :AUToscale

This command causes the analyzer to evaluate the current input signal and find the optimum conditions for displaying the signal. It adjusts the vertical gain and offset for the channel, and sets the time base on the lowest numbered input channel that has a signal.

If signals cannot be found on any vertical input, the analyzer is returned to its former state.

Autoscale sets the following:

- Channel Display, Scale, and Offset
- Trigger and Level
- Time Base Scale and Position

Autoscale turns off the following:

- Measurements on sources that are turned off
- Functions
- Windows
- Memories

No other controls are affected by Autoscale.

Example

This example automatically scales the analyzer for the input signal.

```
10 OUTPUT 707;":AUTOSCALE"
20 END
```

Query

:AUToscale?

Returns a string explaining the results of the last autoscale. The string is empty if the last autoscale completed successfully. The returned string stays the same until the next autoscale is executed.

The following are examples of strings returned by the AUToscale? query.

```
No channels turned on
Left module requires calibration for autoscale
Right module requires calibration for autoscale
Channel n signal is too small
Channel n signal is too high
Channel n offset is too low
Channel n offset is too high
No trigger or trigger too slow
Trigger is in Free Run
Unable to set horizontal scale/delay for channel n
```

Returned Format

[:AUToscale] <string>

BLANk

Command	:BLANk {CHANnel<N> | FUNCtion<N> | WMEMory<N> | RESPonse<N> | HISTogram}
	This command turns off an active channel, function, waveform memory, TDR response or histogram. The VIEW command turns them on.
<N>	An integer, 1 through 4.
Example	This example turns off channel 1.

```
10 OUTPUT 707;":BLANK CHANNEL1"
20 END
```

CDISplay

Command	:CDISplay
	This command clears the display and resets all associated measurements. If the analyzer is stopped, all currently displayed data is erased. If the analyzer is running, all of the data in active channels and functions is erased; however, new data is displayed on the next acquisition. Waveform memories are not erased.
Example	This example clears the analyzer display.

```
10 OUTPUT 707;":CDISPLAY"
20 END
```

COMMents

Command	:COMMents {LMODule | RMODule},"<comments_text>"
	This command sets the comments field for the module. This field is used to describe options included in the module, or for user comments about the module.
<comments>	Represents the ASCII string enclosed in quotation marks.
Example	10 OUTPUT 707;”:COMMENTS LMODULE” 20 END
Query	:COMMents? {LMODule | RMODule}
	The query returns a string with the comments field associated with the module.
Returned Format	[:COMMents] <string>

CREE (Clock Recovery Event Enable Register)

Command

:CREE <mask>

This command sets a mask into the Clock Recovery Event Enable Register.

A “1” in a bit position enables the corresponding bit in the Clock Recovery Event Register to set bit 7 in the Operation Status Register.

<mask>

The decimal weight of the enabled bits. Some of the useful mask values are shown below.

Enable	Mask Value
Block all bits	0
Enable UNLK, block all others	1
Enable LOCK, block all others	2
Enable NSPR1, block all others	4
Enable SPR1, block all others	8
Enable NSPR2, block all others	16
Enable SPR2, block all others	32

Query

:CREE?

The query returns the current decimal value in the Clock Recovery Event Enable Register.

Returned Format

[:CREE] <mask><NL>

CRER? (Clock Recovery Event Register)

Query

:CRER?

This query returns the current value of the Clock Recovery Event Register as a decimal number and also clears this register.

Bit 0 (UNLK) of the Clock Recovery Event Register is set when the clock recovery module becomes unlocked.

Bit 1 (LOCK) of the Clock Recovery Event Register is set when the clock recovery module becomes locked.

Bit 2 (NSPR1) of the Clock Recovery Event Register is set when the clock recovery module transitions to no longer detecting an optical signal on receiver one.

Bit 3 (SPR1) of the Clock Recovery Event Register is set when the clock recovery module transitions to detecting an optical signal on receiver one.

Bit 4 (NSPR2) of the Clock Recovery Event Register is set when the clock recovery module transitions to no longer detecting an optical signal on receiver two.

Bit 5 (SPR2) of the Clock Recovery Event Register is set when the clock recovery module transitions to detecting an optical signal on receiver two.

Returned Format [:CRER] <value><NL>

DIGitize

Command :DIGitize {CHANnel<N> | FUNCtion<N> | RESPonse<N>}

This command invokes a special mode of data acquisition that is more efficient than using the RUN command. This command initializes the selected channels or functions, then acquires them according to the current analyzer settings. When all signals are completely acquired, the analyzer is stopped.

If channel, function, or response parameters are specified, then these are the only waveforms acquired. To speed up acquisition, these waveforms are not displayed and their display state indicates “off.” Subsequent to the digitize operation, the display of the acquired waveforms may be turned on for viewing, if desired. Other sources are turned off and their data is invalidated.

Full Range of Measurement and Math Operators are Available

Even though digitized waveforms are not displayed, the full range of measurement and math operators may be performed on them.

If you use the DIGitize command with no parameters, the digitize operation is performed on the channels or functions that were acquired with a previous digitize, run, or single operation. In this case, the display state of the acquired waveforms is not changed. Because the command executes more quickly without parameters, this form of the command is useful for repetitive measurement sequences. You can also use this mode if you want to view the digitize results because the display state of the digitized waveforms is not affected.

Data acquired with the DIGitize command is placed in the normal channel, function, or response, just as when it is acquired with the RUN command.

See Chapter 6, “Sample Programs” for examples of how to use DIGitize and its related commands.

<N> An integer, 1 through 4.

Example This example acquires data on channel 1 and function 2.

```
10 OUTPUT 707;":DIGITIZE CHANNEL1,FUNCTION2"
20 END
```

The ACQuire subsystem commands set up conditions such as TYPE and COUNT for the next DIGitize command.

The WAVeform subsystem commands determine how the data is transferred out of the analyzer, and how to interpret the data.

LER? (Local Event Register)

Query :LER?

This query reads the Local (LCL) Event Register. A “1” is returned if a remote-to-local transition has taken place due to the front-panel Local key being pressed. A “0” is returned if a remote-to-local transition has not taken place.

Returned Format [:LER] {1 | 0}<NL>

Example The following example checks to see if a remote-to-local transition has taken place and places the result in the string variable, Answer$, and then prints the result to the controller’s screen.

```
10 Dim Answer$[50]            !Dimension variable
20 OUTPUT 707;":LER?"
30 ENTER 707; Answer$
40 PRINT Answer$
50 END
```

After the LCL Event Register is read, it is cleared.

Once this bit is set, it can only be cleared by reading the Status Byte, reading the register with the LER? query, or sending a *CLS common command.

LTEE (Limit Test Event Enable register)

Command :LTEE <mask>

This command sets a mask into the Limit Test Event Enable register.

A “1” in a bit position enables the corresponding bit in the Limit Event Register to set bit 8 in the Operation Status Register.

<mask> The decimal weight of the enabled bits. Only bits 0 and 1, of the Limit Test Event Register, are used at this time. The useful mask values are shown in the following table.

Enable	Mask Value
Block COMP and FAIL	0
Enable COMP, block FAIL	1
Enable FAIL, block COMP	2
Enable COMP and FAIL	3

Query :LTEE?

The query returns the current decimal value in the Limit Test Event Enable Register.

Returned Format [:LTEE] <mask><NL>

LTER? (Limit Test Event Register)

Query :LTER?

This query returns the current value of the Limit Test Event Register as a decimal number and also clears this register.

Bit 0 (COMP) of the Limit Test Event Register is set when the Limit Test completes. The Limit Test completion criteria are set by the LTESt:RUN command.

Bit 1 (FAIL) of the Limit Test Event Register is set when the Limit Test fails. Failure criteria for the Limit Test are defined by the LTESt:FAIL command.

Returned Format [:LTER] <value><NL>

MODel?

Query :MODel? {FRAMe | LMODule | RMODule}

This query returns the Agilent model number for the analyzer frame or module.

Returned Format [:MODel] <string>

<string> A six-character alphanumeric model number in quotation marks. Output is determined by header and longform status as in Table 8-1.

Table 8-1. Model? Returned Format

HEADER		LONGFORM		RESPONSE
ON	OFF	ON	OFF	
	X		X	86100A
	X	X		86100A
X			X	:MOD 86100A
X		X		:MODEL 86100A

Example

This example places the model number of the frame in a string variable, Model$, then prints the contents of the variable on the computer's screen.

```
10 Dim Model$[13]                    !Dimension variable
20 OUTPUT 707;":Model? FRAME"
30 ENTER 707; Model$
40 PRINT Model$
50 END
```

MTEE (Mask Test Event Enable Register)

Command

:MTEE <mask>

This command sets a mask into the Mask Event Enable register.

A “1” in a bit position enables the corresponding bit in the Mask Test Event Register to set bit 10 in the Operation Status Register.

<mask>

The decimal weight of the enabled bits. Only bits 0 and 1, of the Mask Test Event Register, are used at this time. The useful mask values are shown in the following table.

Enable	Mask Value
Block COMP and FAIL	0
Enable COMP, block FAIL	1
Enable FAIL, block COMP	2
Enable COMP and FAIL	3

Query

:MTEE?

The query returns the current decimal value in the Mask Event Enable Register.

Returned Format

[:MTEE] <mask><NL>

MTER? (Mask Test Event Register)

Query :MTER?

This query returns the current value of the Mask Test Event Register as a decimal number and also clears this register.

Bit 0 (COMP) of the Mask Test Event Register is set when the Mask Test completes.

Bit 1 (FAIL) of the Mask Test Event Register is set when the Mask Test fails. This will occur whenever any sample is recorded within any region defined in the mask.

Returned Format [:MTER] <value><NL>

OPEE

Command :OPEE <mask>

This command sets a mask in the Operation Status Enable register. Each bit that is set to a “1” enables that bit to set bit 7 in the Status Byte Register, and potentially causes an SRQ to be generated. Bit 5, Wait for Trig, is used. Other bits are reserved.

<mask> The decimal weight of the enabled bits.

Query :OPEE?

The query returns the current value contained in the Operation Status Enable register as a decimal number.

Returned Format [:OPEE] <value><NL>

OPER?

Query :OPER?

This query returns the value contained in the Operation Status Register as a decimal number and also clears this register. This register is the summary of the CLCK bit (bit 7), LTEST bit (bit 8), ACQ bit (bit 9) and MTEST bit (bit 10).

The CLCK bit is set by the Clock Recovery Event Register and indicates that a clock event has occurred. The LTEST bit is set by the Limit Test Event Register and indicates that a limit test has failed or completed. The ACQ bit is set by the Acquisition Event Register and indicates that an acquisition limit test has completed. The MTEST bit is set by the Mask Test Event Register and indicates that a mask limit test has failed or completed.

Returned Format [:OPER] <value><NL>

PRINt

Command :PRINt

This command outputs a copy of the screen to a printer or other device destination specified in the HARDcopy subsystem. You can specify the selection of the output and the printer using the HARDcopy subsystem commands.

Example This example outputs a copy of the screen to a printer or a disk file.

```
10 OUTPUT 707;":PRINT"
20 END
```

RECall:SETup

Command :RECall:SETup <setup_memory_num>

This command recalls a setup that was saved in one of the analyzer's setup memories. You can save setups using either the STORe:SETup command or the front panel.

<setup_memory_num> Setup memory number, an integer, 0 through 9.

Example This command recalls a setup from setup memory 2.

```
10 OUTPUT 707;":RECall:SETup 2"
20 END
```

RUN

Command :RUN

This command starts the analyzer running. When the analyzer is running, it acquires waveform data according to its current settings. Acquisition runs repetitively until the analyzer receives a STOP command.

Example This example causes the analyzer to acquire data repetitively.

```
10 OUTPUT 707;":RUN"
20 END
```

SERial (Serial Number)

Command :SERial {FRAMe | LMODule | RMODule},<string>

This command sets the serial number for the analyzer frame or module. The serial number is entered by Agilent Technologies. Therefore, setting the serial number is not normally required unless the analyzer is serialized for a different application.

<string> A ten-character alphanumeric serial number enclosed with quotation marks.

The analyzer's serial number is part of the string returned for the *IDN? query, described in Chapter 7, "Common Commands".

Example This example sets the serial number for the analyzer's frame to "1234A56789".

```
10 OUTPUT 707;":SERIAL FRAME,""1234A56789"""
20 END
```

Query :SERial? {FRAMe | LMODule | RMODule}

The query returns the current serial number string for the specified frame or module.

Returned Format [:SERial] <string><NL>

Example

```
10 Dim Serial$[50]                  !Dimension variable
20 OUTPUT 707;":SERIAL? FRAME"
30 ENTER 707; Serial$
40 PRINT SERIAL$
50 END
```

SINGle

Command :SINGle

This command causes the analyzer to make a single acquisition when the next trigger event occurs. It should be followed by *WAI, *OPC, or *OPC? in order to synchronize data acquisition with remote control.

Example This example sets up the analyzer to make a single acquisition when the next trigger event occurs.

```
10 OUTPUT 707;":SINGLE"
20 END
```

STOP

Command :STOP

This command causes the analyzer to stop acquiring data for the active display. To restart the acquisition, use the RUN or SINGle command.

Example This example stops the current data acquisition.

```
10 OUTPUT 707;":STOP"
20 END
```

STORe:SETup

Command :STORe:SETup <setup_memory_num>

This command saves the current analyzer setup in one of the setup memories.

<setup_memory_num> Setup memory number, an integer, 0 through 9.

STORe:WAVEform

Command :STORe:WAVEform {CHANnel<N> | FUNCtion<N> | WMEMory<N> | RESPonse<N>},{WMEMory<N>}

This command copies a channel, function, stored waveform, or TDR response to a waveform memory. The parameter preceding the comma specifies the source and can be any channel, function, response or waveform memory. The parameter following the comma is the destination, and can be any waveform memory.

<N> An integer, 1 through 4.

Example This example copies channel 1 to waveform memory 3.

```
10 OUTPUT 707;":STORE:WAVEFORM CHANNEL1,WMEMORY3"
20 END
```

TER? (Trigger Event Register)

Query :TER?

This query reads the Trigger Event Register. A “1” is returned if a trigger has occurred. A “0” is returned if a trigger has not occurred.

Returned Format [:TER] {1 | 0}<NL>

Example

This example checks the current status of the Trigger Event Register and places the status in the string variable, Current$, then prints the contents of the variable to the computer's screen.

```
10 DIM Current$[50]          !Dimension variable
20 OUTPUT 707;":TER?"
30 ENTER 707;Current$
40 PRINT Current$
50 END
```

Once this bit is set, you can clear it only by reading the register with the TER? query, or by sending a *CLS common command. After the Trigger Event Register is read, it is cleared.

UEE (User Event Enable register)

Command

:UEE <mask>

This command sets a mask into the User Event Enable register. A “1” in a bit position enables the corresponding bit in the User Event Register to set bit 1 in the Status Byte Register and, thereby, potentially cause an SRQ to be generated. Only bit 0 of the User Event Register is used at this time; all other bits are reserved.

<mask>

The decimal weight of the enabled bits.

Query

:UEE?

The query returns the current decimal value in the User Event Enable register.

Returned Format

[:UEE] <mask><NL>

UER? (User Event Register)

Query

:UER?

This query returns the current value of the User Event Register as a decimal number and also clears this register. Bit 0 (LCL - Remote/Local change) is used. All other bits are reserved.

Returned Format

[:UER] <value><NL>

VIEW

Command :VIEW {CHANnel<N> | FUNCtion<N> | WMEMory<N> | RESPonse<N> | HISTogram}

This command turns on a channel, function, waveform memory, TDR response, or histogram.

<N> An integer, 1 through 4.

Example This example turns on channel 1.

```
10 OUTPUT 707;":VIEW CHANNEL1"
20 END
```

See Also The BLANk command turns off a channel, function, waveform memory, TDR response, or histogram.

System Commands

System Commands

SYSTem subsystem commands control the way in which query responses are formatted, send and receive setup strings, and enable reading and writing to the advisory line of the analyzer. You can also set and read the date and time in the analyzer using the SYSTem subsystem commands.

DATE

Command :SYSTem:DATE <day>,<month>,<year>

This command sets the date in the analyzer, and is not affected by the *RST common command.

<day> Specifies the day in the format <1. . . .31>.

<month> Specifies the month in the format <1, 2, ...12> | <JAN, FEB, MAR . . . .>.

<year> Specifies the year in the format <yyyy> | <yy>. The values range from 1992 to 2035.

Example The following example sets the date to July 1, 1997.

```
10 OUTPUT 707;":SYSTEM:DATE 7,1,97"
20 END
```

Query :SYSTem:DATE?

The query returns the current date in the analyzer.

Returned Format [:SYSTem:DATE] <day> <month> <year>><NL>

Example The following example queries the date.

```
10 DIM Date$ [50]
20 OUTPUT 707;":SYSTEM:DATE?"
30 ENTER 707; Date$
40 PRINT Date$
```

DSP

Command :SYSTem:DSP <string>

This command writes a quoted string, excluding quotation marks, to the advisory line of the instrument display. If you want to clear a message on the advisory line, send a null (empty) string.

<string> An alphanumeric character array up to 92 bytes long.

Example The following example writes the message, “Test 1” to the advisory line of the analyzer.

```
10 OUTPUT 707;":SYSTEM:DSP ""Test 1"""
20 END
```

Query :SYSTem:DSP?

The query returns the last string written to the advisory line. This may be a string written with a SYSTem:DSP command, or an internally generated advisory.

The string is actually read from the message queue. The message queue is cleared when it is read. Therefore, the displayed message can only be read once over the bus.

Returned Format [:SYSTem:DSP] <string><NL>

Example The following example places the last string written to the advisory line of the analyzer in the string variable, Advisory$. Then, it prints the contents of the variable to the controller's screen.

```
10 DIM Advisory$[89]                !Dimension variable
20 OUTPUT 707;":SYSTEM:DSP?"
30 ENTER 707;Advisory$
40 PRINT Advisory$
50 END
```

ERRor?

Query :SYSTem:ERRor? [{NUMBer | STRing}]

This query outputs the next error number in the error queue over the GPIB. When either NUMBer or no parameter is specified in the query, only the numeric error code is output. When STRing is specified, the error number is output followed by a comma and a quoted string describing the error. Table 9-1 on page 9-5 lists the error numbers and their corresponding error messages. The error messages are also listed in Chapter 29, “Error Messages”, where possible causes are given for each message.

Returned Format [:SYSTem:ERRor] <error_number>[,<quoted_string>]<NL>

<error_number> A numeric error code.

<quoted_string> A quoted string describing the error.

Example The following example reads the oldest error number and message in the error queue into the string variable, Condition$, then prints the contents of the variable to the controller's screen.

```
10 DIM Condition$[64]               !Dimension variable
20 OUTPUT 707;":SYSTEM:ERROR? STRING"
30 ENTER 707;Condition$
40 PRINT Condition$
50 END
```

This analyzer has an error queue that is 30 errors deep and operates on a first-in, first-out (FIFO) basis. Successively sending the SYSTem:ERRor query returns the error numbers in the order that they occurred until the queue is empty. When the queue is empty, this query returns headers of 0, “No error.” Any further queries return zeros until another error occurs. Note that front-panel generated errors are also inserted in the error queue and the Event Status Register.

Send *CLS Before Other Commands or Queries

Send the *CLS common command to clear the error queue and Event Status Register before you send any other commands or queries.

See Also

Chapter 29, “Error Messages” for more information on error messages and their possible causes.

Table 9-1. Error Messages

Error Number	Description	Error Number	Description
0	No error	–158	String data not allowed
–100	Command error	–160	Block data error
–101	Invalid character	–161	Invalid block data
–102	Syntax error	–168	Block data not allowed
–103	Invalid separator	–170	Expression error
–104	Data type error	–171	Invalid expression
–105	GET not allowed	–178	Expression data not allowed
–108	Parameter not allowed	–200	Execution error
–109	Missing parameter	–222	Data out of range
–112	Program mnemonic too long	–223	Too much data
–113	Undefined header	–224	Illegal parameter value
–121	Invalid character in number	–310	System error
–123	Numeric overflow	–350	Too many errors
–124	Too many digits	–400	Query error
–128	Numeric data not allowed	–410	Query INTERRUPTED
–131	Invalid suffix	–420	Query UNTERMINATED
–138	Suffix not allowed	–430	Query DEADLOCKED
–141	Invalid character data	–440	Query UNTERMINATED after indefinite response
–144	Character data too long		

HEADer

Command

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

This command specifies whether the instrument will output a header for query responses. When SYSTem:HEADer is set to ON, the query responses include the command header.

Example

The following example sets up the analyzer to output command headers with query responses.

```
10 OUTPUT 707;":SYSTEM:HEADER ON"
20 END
```

Query

```
:SYSTem:HEADer?
```

The query returns the state of the SYSTem:HEADer command.

Returned Format

```
[:SYSTem:HEADer] {1 | 0}<NL>
```

Example

This example examines the header to determine the size of the learn string. Memory is then allocated to hold the learn string before reading it. To output the learn string, the header is sent, then the learn string and the EOF.

```
10 DIM Header$[64]
20 OUTPUT 707;"syst:head on"
30 OUTPUT 707;":syst:set?"
40 More_chars:  !
50 ENTER 707 USING "#,A";This_char$
60 Header$=Header$&This_char$
70 IF This_char$<>"#" THEN More_chars
80 !
90 ENTER 707 USING "#,D";Num_of_digits
100 ENTER 707 USING "#,"&VAL$(Num_of_digits)&"D";Set_size
110 Header$=Header$&"#"&VAL$(Num_of_digits)&VAL$(Set_size)
120!
130 ALLOCATE INTEGER Setup(1:Set_size)
140 ENTER 707 USING "#,B";Setup(*)
150 ENTER 707 USING "#,A";Eof$
160 !
170 OUTPUT 707 USING "#,-K";Header$
180 OUTPUT 707 USING "#,B";Setup(*)
190 OUTPUT 707 USING "#,A";Eof$
200
```

Turn Headers Off when Returning Values to Numeric Variables

Turn headers off when returning values to numeric variables. Headers are always off for all common command queries because headers are not defined in the IEEE 488.2 standard.

LONGform

Command

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

This command specifies the format for query responses. If the LONGform is set to OFF, command headers and alpha arguments are sent from the instrument in the short form (abbreviated spelling). If LONGform is set to ON, the whole word is output.

This command has no effect on input headers and arguments sent to the instrument. Headers and arguments may be sent to the instrument in either the long form or short form, regardless of the current state of the LONGform command.

Example

The following example sets the format for query response from the instrument to the short form (abbreviated spelling).

```
10 OUTPUT 707;":SYSTEM:LONGFORM OFF"
20 END
```

Query

```
:SYSTem:LONGform?
```

The query returns the current state of the SYSTem:LONGform command.

Returned Format

```
[:SYSTem:LONGform] {0 | 1}<NL>
```

Example

The following example checks the current format for query responses from the oscilloscope and places the result in the string variable, Result$, then prints the contents of the variable to the controller’s screen.

```
10 DIM Result$[50]                          !Dimension variable
20 OUTPUT 707;":SYSTEM:LONGFORM?"
30 ENTER 707;Result$
40 PRINT Result$
50 END
```

MODE

Command :SYSTem:MODE {EYE | OSCilloscope | TDR}

This command sets the system mode.

Example The following example sets the instrument mode to Eye/Mask mode.

```
10 OUTPUT 707;":SYSTEM:MODE OSCilloscope"
20 END
```

Query :SYSTem:MODE?

The query returns the current state of the SYSTem:MODE command.

Returned Format [:SYSTem:MODE] {EYE | OSCilloscope | TDR}

Example The following example checks the current instrument mode of the analyzer, and places the result in the string variable, Result$. Then, it prints the contents of the variable to the controller's screen.

```
10 DIM Result$[50]                    !Dimension variable
20 OUTPUT 707;":SYSTEM:MODE?"
30 ENTER 707;Result$
40 PRINT Result$
50 END
```

SETup

Command :SYSTem:SETup <binary_block_data>

This command sets up the instrument as defined by the data in the setup string from the controller.

<binary_block_data> A string, consisting of bytes of setup data. The number of bytes is a dynamic number that is read and allocated by the analyzer's software.

Example

The following example sets up the instrument as defined by the setup string stored in the variable, Set$.

```
10 OUTPUT 707 USING "#,-K";":SYSTEM:SETUP ";Set$
20 END
```

HP BASIC Image Specifiers

# is an HP BASIC image specifier that suppresses the automatic output of the EOI sequence following the last output item.

K is an HP BASIC image specifier that outputs a number or string in standard form with no leading or trailing blanks.

Query

:SYSTem:SETup?

The query outputs the instrument's current setup to the controller in binary block data format as defined in the IEEE 488.2 standard.

Returned Format

[:SYSTem:SETup] #NX...X<setup data string><NL>

The first character in the setup data string is a number added for disk operations.

Example

The following example stores the current instrument setup in the string variable, Set$.

```
10 DIM Set$[15000]                        !Dimension variable
20 OUTPUT 707;":SYSTEM:HEADER OFF"        !Response headers off
30 OUTPUT 707;":SYSTEM:SETUP?"
40 ENTER 707 USING "-K";Set$
50 END
```

HP BASIC Image Specifiers

–K is an HP BASIC image specifier which places the block data in a string, including carriage returns and line feeds, until EOI is true, or when the dimensioned length of the string is reached.

SYSTem:SETup Can Operate Just Like *LRN

When headers and LONGform are on, the SYSTem:SETup query operates the same as the *LRN query in the common commands. Otherwise, *LRN and SETup are not interchangeable.

TIME

Command	:SYSTem:TIME <hour>,<minute>,<second>
	This command sets the time in the instrument, and is not affected by the *RST common command.
<hour>	0. . . .23
<minute>	0. . . .59
<second>	0. . . .59

Example

```
10 OUTPUT 707;":SYSTEM:TIME 10,30,45"
20 END
```

Query :SYSTem:TIME?

The query returns the current time in the instrument.

Returned Format [:SYSTem:TIME] <hour>,<minute>,<second>

10

Acquire Commands

Acquire Commands

The ACQuire subsystem commands set up conditions for acquiring waveform data, including the DIGitize root level command. The commands in this subsystem select the type of data, the number of averages, and the number of data points. This subsystem also includes commands to set limits on how much data is acquired, and specify actions to execute when acquisition limits are met.

AVERage

Command :ACQuire:AVERage {{ON | 1} | {OFF | 0}}

This command enables or disables averaging. When ON, the analyzer acquires multiple data values for each time bucket, and averages them. When OFF, averaging is disabled. To set the number of averages, use the :ACQuire:COUNt command described later in this chapter.

Averaging is not available in PDETect mode.

Example This example turns averaging on.

```
10 OUTPUT 707;":ACQUIRE:AVERAGE ON"
20 END
```

Query :ACQuire:AVERage?

Returned Format [:ACQuire:AVERage] {1 | 0}<NL>

BEST

Command :ACQuire:BEST {THRuput | FLATness}

When averaging is enabled with ACQuire:AVERage, the FLATness option improves the step flatness by using a signal processing algorithm within the instrument. You should use this option when performing TDR measurements or when step flatness is important. The THRuput option improves the instrument's throughput and should be used whenever best flatness is not required.

Example The following example sets the instrument to best step flatness.

```
10 OUTPUT 707;":ACQUIRE:BEST FLATNESS"
20 END
```

Query :ACQuire:BEST?

The query returns the current acquisition algorithm setting.

Returned Format [:ACQuire:BEST] {THRuput | FLATness}<NL>

Example The following example obtains the current setting of the acquisition algorithm from the instrument, stores it in the variable, Best$, then prints the contents of the variable to the controller's screen.

```
10 DIM Best$[50]                         !Define variable
20 OUTPUT 707;":ACQUIRE:BEST?"
30 ENTER 707;Best$
40 PRINT Best$
50 END
```

COUNt

Command :ACQuire:COUNt <value>

This command sets the number of averages for the waveforms. In the AVERage mode, the ACQuire:COUNt command specifies the number of data values to be averaged for each time bucket before the acquisition is considered complete for that time bucket.

<value> An integer, 1 to 4096, specifying the number of data values to be averaged.

Example The following example specifies that 16 data values must be averaged for each time bucket to be considered complete.

```
10 OUTPUT 707;":ACQUIRE:COUNT 16"
20 END
```

Query :ACQuire:COUNt?

The query returns the currently selected count value.

Returned Format [:ACQuire:COUNt] <value><NL>

<value> An integer, 1 to 4096, specifying the number of data values to be averaged.

Example The following example checks the currently selected count value and places that value in the string variable, Result$. Then the program prints the contents of the variable to the controller's screen.

```
10 DIM Result$[50]                    !Dimension variable
20 OUTPUT 707;":ACQUIRE:COUNT?"
30 ENTER 707;Result$
40 PRINT Result$
50 END
```

POINts

Command :ACQuire:POINts {AUTO | <points_value>}

This command sets the requested memory depth for an acquisition. Always query the points value with the WAVeform:POINts query or WAVeform:PRE-amble to determine the actual number of acquired points.

You can set the points value to AUTO, which allows the analyzer to select the number of points based upon the sample rate and time base scale.

<points_value> An integer representing the memory depth. The points value range is 16 to 4096 points.

Example This example sets the memory depth to 500 points.

```
10 OUTPUT 707;":ACQUIRE:POINTS 500"
20 END
```

Query :ACQuire:POINts?

The query returns the requested memory depth.

Returned Format [:ACQuire:POINts] <points_value><NL>

Example This example checks the current setting for memory depth and places the result in the string variable, Length$. Then the program prints the contents of the variable to the controller's screen.

```
10 DIM Length$[50]                    !Dimension variable
20 OUTPUT 707;":ACQUIRE:POINTS?"
30 ENTER 707;Length$
40 PRINT Length$
50 END
```

See Also :WAVeform:DATA

RUNTil

Command	:ACQuire:RUNTil {OFF | WAVeforms,<number_of_waveforms> | SAMples, <number_of_samples>}
	This command selects the acquisition run until mode. The acquisition may be set to run until *n* waveforms or *n* samples have been acquired, or to run forever (OFF).If more than one run until criteria is set, then the instrument will act upon the completion of whichever run until criteria is achieved first.
<number_of_ waveforms	An integer from 1 to 2^{31}–1.
<number_of_samples>	An integer from 1 to 2^{31}–1.
Example	The following example specifies that the acquisition runs until 200 samples have been obtained.

```
10 OUTPUT 707;":ACQuire:RUNTIL SAMPLES,200"
20 END
```

Query	:ACQuire:RUNTil?
	The query returns the currently selected run until state.
Returned Format	[:ACQuire:RUNTil] {OFF | WAVeform, <n waveforms> | SAMPles, <n samples>}<NL>
Example	The following example returns the result of the run until query and prints it to the controller's screen.

```
10 DIM Runt$[50]
20 OUTPUT 707;":ACQuire:RUNTIL?"
30 ENTER 707;Runt$
40 PRINT Runt$
50 END
```

SSCReen

Command	:ACQuire:SSCReen {OFF | DISK [,<filename>]}
	This command saves a copy of the screen when the acquisition limit is reached.
OFF	Turns off the save action.
DISK	A different set of commands is provided to control the print to disk.
<filename>	An ASCII string enclosed in quotation marks. If no filename is specified, a default filename is assigned. This filename will be *AcqLimitScreenX*, where X is an incremental number assigned by the instrument.

If a filename is specified without a path, the default path will be C:\User Files\screen images. The default file type is a bitmap (.bmp). The following graphics formats are available by specifying a file extension: PCX files (.pcx), EPS files (.eps), Postscript files (.ps) and GIF files (.gif).

Example The following example saves a copy of the screen to the disk when acquisition limit is reached. Additional disk-related controls are set using the SSCReen:AREA and SSCReen:IMAGe commands.

```
10 OUTPUT 707;":ACQUIRE:SSCREEN DISK"
20 END
```

Query :ACQuire:SSCReen?

The query returns the current state of the SSCReen command.

Returned Format [:ACQuire:SSCReen] {OFF | DISK [,<filename>]}<NL>

Example The following example returns the destination of the save screen when acquisition limit is reached and prints the result to the controller's screen.

```
10 DIM SSCR$[50]
20 OUTPUT 707;":ACQUIRE:SSCREEN?"
30 ENTER 707;SSCR$
40 PRINT SSCR$
50 END
```

SSCReen:AREA

Command :ACQuire:SSCReen:AREA {GRATicule | SCReen}

This command selects which data from the screen is to be saved to disk when the run until condition is met. When you select GRATicule, only the graticule area of the screen is saved (this is the same as choosing Waveforms Only in the Specify Report Action for acquisition limit test dialog box). When you select SCReen, the entire screen is saved.

Example This example selects the graticule for saving.

```
10 OUTPUT 707;":ACQUIRE:SSCREEN:AREA GRATICULE"
20 END
```

Query :ACQuire:SSCReen:AREA?

The query returns the current setting for the area of the screen to be saved.

Returned Format [:ACQuire:SSCReen:AREA] {GRATicule | SCReen}<NL>

Example

This example places the current selection for the area to be printed in the string variable, Selection$, then prints the contents of the variable to the computer's screen.

```
10 DIM Selection$[50]                    !Dimension variable
20 OUTPUT 707;":ACQUIRE:SSCREEN:AREA?"
30 ENTER 707;Selection$
40 PRINT Selection$
50 END
```

SSCReen:IMAGe

Command

```
:ACQuire:SSCReen:IMAGe {NORMal | INVert | MONochrome}
```

This command saves the screen image to disk normally, inverted, or in monochrome. IMAGe INVert is the same as choosing Invert Background Waveform Color in the Specify Report Action for acquisition limit test dialog box.

Example

This example sets the image output to normal.

```
10 OUTPUT 707;":ACQuire:SSCReen:IMAGE NORMAL"
20 END
```

Query

```
:ACQuire:SSCReen:IMAGe?
```

The query returns the current image setting.

Returned Format

```
[:ACQuire:SSCReen:IMAGe] {NORMal | INVert | MONochrome}<NL>
```

Example

This example places the current setting for the image in the string variable, Setting$, then prints the contents of the variable to the computer's screen.

```
10 DIM Setting$[50]                              !Dimension variable
20 OUTPUT 707;":ACQUIRE:SSCREEN:IMAGE?"
30 ENTER 707;Setting$
40 PRINT Setting$
50 END
```

SWAVeform

Command

```
:ACQuire:SWAVeform <source>, <destination>,[<filename>[, <format>]]
```

This command saves waveforms from a channel, function, TDR response, or waveform memory in the event of a failure detected by the limit test. Each waveform source can be individually specified, allowing multiple channels, responses, or functions to be saved to disk or waveform memories. Setting a particular source to OFF removes any waveform save action from that source.

<source>

```
{CHANnelN | FUNCtionN | WMEMoryN | RESPonseN}
```

<destination> {OFF | WMEMoryN | DISK}

<filename> An ASCII string enclosed in quotes. If no filename is specified, a default filename will be assigned. The default filenames will be *AcqLimitChN_X*, *AcqLimitFnN_X*, *AcqLimitMemN_X* or *AcqLimitRspN_X*, where X is an incremental number assigned by the instrument.

If a specified filename contains no path, the default path will be C:\User Files\waveforms.

<format> {TEXT [,YVALues | VERBose] | INTernal}

Where INTernal is the default format, and VERBose is the default format for TEXT.

Example The following example turns off the saving of waveforms from channel 1.

```
10 OUTPUT 707;":ACQUIRE:SWAVEFORM CHAN1,OFF"
20 END
```

Query :ACQuire:SWAVeform? <source>

The query returns the current state of the :ACQuire:SWAVeform command.

Returned Format [:ACQuire:SWAVeform]<source>, <destination>, [<filename>[,<format>]]<NL>

Example The following example returns the current parameters for saving waveforms.

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

SWAVeform:RESet

Command :ACQuire:SWAVeform:RESet

This command sets the save destination for all waveforms to OFF. Setting a source to OFF removes any waveform save action from that source. This is a convenient way to turn off all saved waveforms if it is unknown which are being saved.

Example

```
10 OUTPUT 707;":ACQuire:SWAVeform:RESet"
20 END
```

11

Calibration Commands

Calibration Commands

This section briefly explains the calibration of the 86100A digital communications analyzer. It is intended to give you and the calibration lab personnel an understanding of the calibration procedure and how the calibration subsystem is intended to be used. Also, this section acquaints you with the terms used in this chapter, help screens, and data sheets.

A calibration procedure is included at the end of this chapter.

Mainframe Calibration

Mainframe calibration establishes calibration factors for the analyzer. These factors are stored in the analyzer's hard disk. You initiate the calibration from the Calibration menu or by sending the :CALibrate:FRAMe:STARt command.

You should calibrate the analyzer mainframe periodically (at least annually), or if the ambient temperature since the last calibration has changed more than ±5°C. The temperature change since the last calibration is shown on the calibration status screen which is found under the Horizontal (time base) tab on the “Calibrate/All Calibrations” dialog. It is the line labeled:

Cd ΔT ____________ °C.

See Also The Service Guide has more details about the mainframe calibration.

Module Calibration

You initiate a module calibration from the Vertical (amplitude) tab on the “Calibrate/All Calibrations” dialog or by sending the :CALibrate:MODule:VERTical command.

The vertical (amplitude) calibration, also referred to as the module calibration, is used to enhance the measurement precision of the instrument. It is recommended you routinely perform this calibration for best measurement accuracy.

When a vertical calibration is performed, the instrument establishes calibration factors for the module. The calibration factors compensate for imperfections in the measurement system, such as variations due to the ambient temperature. This results in the best instrument precision. The module calibration factors are valid only for the mainframe and slot in which the module was calibrated. You can install the module in the slots provided for Channels 1 and 2, or for Channel 3 and 4.

Note

In order for the calibration to be accurate the temperature of the module must reach equilibrium prior to performing the calibration.

The vertical calibration is self-contained so the instrument does not require an external equipment setup. In fact, the instrument will display a message box instructing you to remove or disable all inputs to the module to be calibrated. The duration of the calibration is typically between 60 and 90 seconds.

A vertical calibration is recommended when:

- the instrument power has been cycled
- a module has been removed and then reinserted since the last calibration
- a change in the temperature of the mainframe exceeds 1°C compared to the temperature of the last vertical calibration ($\Delta T > 1°C$)
- The time since the last calibration has exceeded 10 hours

Note

Reseating the module into the mainframe can affect the electrical connections, which in turn can affect the calibration accuracy.

Note

A positive value for Δ/T indicates how many degrees warmer the current mainframe temperature is compared to the temperature of the mainframe at the time of the last module vertical calibration.

CAUTION

The input circuits can be damaged by electrostatic discharge (ESD). Avoid applying static discharges to the front-panel input connectors. Momentarily short the center and outer conductors of coaxial cables *prior* to connecting them to the front-panel inputs. *Before* touching the front-panel input connectors be sure to first touch the frame of the instrument. Be sure the instrument is properly earth-grounded to prevent buildup of static charge. Wear a wrist-strap or heel-strap.

Probe Calibration

The probe calibration is initiated from the Probe tab on the “Calibrate/All Calibrations” dialog or by sending either the :CALibrate:PROBe command or the :CHANnel<N>:PROBe:CALibrate command.

The probe calibration allows the instrument to identify the offset and the gain, or loss, of specific probes that are connected to an electrical channel of the instrument. Those factors are then applied to the calibration of that channel. The instrument calibrates the vertical scale and offset based on the voltage measured at the tip of the probe or the cable input.

Note

For passive or non-identified probes, the instrument adjusts the vertical scale factors only if a probe calibration is performed.

Typically probes have standard attenuation factors, such as divide by 10, divide by 20, or divide by 100. If the probe being calibrated has a non-standard attenuation, the instrument will adjust the vertical scale factors of the input channel to match this attenuation.

CAUTION

The input circuits can be damaged by electrostatic discharge (ESD). Avoid applying static discharges to the front-panel input connectors. Momentarily short the center and outer conductors of coaxial cables *prior* to connecting them to the front-panel inputs. *Before* touching the front-panel input connectors be sure to first touch the frame of the instrument. Be sure the instrument is properly earth-grounded to prevent buildup of static charge. Wear a wrist-strap or heel-strap.

Calibration Commands

The commands in the CALibration subsystem initiate the analyzer calibration over GPIB.

Let the Analyzer Warm Up First

Let the analyzer warm up for at least 30 minutes before you calibrate it.

CANCel

Command :CALibrate:CANCel

This command cancels normalization when a calibration message box prompt is displayed.

Example This example cancels the analyzer calibration.

```
10 OUTPUT 707;":CALIBRATE:CANCEL"
20 END
```

CONTinue

Command :CALibrate:CONTinue

This command continues normalization when a calibration message box prompt is displayed.

Example This example continues the analyzer calibration.

```
10 OUTPUT 707;":CALIBRATE:CONTINUE"
20 END
```

ERATio:DLEVel? CHANnel<N>

Query :CALibrate:ERATio:DLEVel? CHANnel<N>

<N> An integer, from 1 to 4.

This query returns the dark level value for the specified channel. If an extinction ratio calibration has been performed the returned value is the calibration result. If no calibration has been performed the default value of 0.0 is returned.

Returned Format	[:CALibrate:ERATio:DLEVel] <value><NL>

ERATio:STARt CHANnel<N>

Command	:CALibrate:ERATio:STARt CHANnel<N>
	This command starts an extinction ratio calibration.
<N>	An integer, from 1 to 4.

ERATio:STATus? CHANnel<N>

Query	:CALibrate:ERATio:STATus? CHANnel<N>
	This query returns `CALIBRATED` or `DEFAULTED`.
Returned Format	[:CALibrate:ERATio:STATus] {CALIBRATED | DEFAULTED}<NL>

FRAMe:LABel

Command	:CALibrate:FRAMe:LABel <label>
	This command is intended for user notes, such as name/initials of the calibrator or special notes about the calibration. It accepts a string of up to 80 characters. The information is optional.
<label>	A string, enclosed with quotes, with a maximum of 80 characters.
Query	:CALibrate:FRAMe:LABel?
	The query returns the currently defined label for the frame.
Returned Format	[:CALibrate:FRAMe:LABel] <quoted string><NL>

FRAMe:STARt

Command	:CALibrate:FRAMe:STARt
	This command starts the annual calibration on the instrument mainframe.

FRAMe:TIME?

Query :CALibrate:FRAMe:TIME?

This query returns the date, time and temperature at which the last full frame calibration process was completed.

Returned Format [:CALibrate:FRAMe:TIME] <time> <NL>

<time> Is in the format: DD MMM YY HH:MM <delta_temp>

<delta_temp> Is the difference between the current temperature and the temperature when the last calibration was done. For example, <delta_temp> might be:

```
–5C
10C
–12C
```

MODule:OCONversion?

Query :CALibrate:MODule:OCONversion? {LMODule | RModule},{WAVelength 1 | WAVelength 2 | USER}

This query returns the optical conversion (responsivity) of the specified channel at the specified wavelength. Wavelength 1 and Wavelength 2 are for factory-calibrated wavelengths. USER is the result of a user optical calibration.

Returned Format [:CALibrate:MODule:OCONversion] <value><NL>

MODule:OPOWer

Command :CALibrate:MODule:OPOWer <optical_power_value>

This command sets the optical power level for an optical channel module calibration. This command should only be used for modules with an optical channel.

Example

```
10 OUTPUT 707;":CALIBRATE:MODULE:OPOWER 500E–6"
20 END
```

MODule:OPTical

Command

:CALibrate:MODule:OPTical {CHAN1 | CHAN2 | CHAN3 | CHAN4}

This command initiates an O/E calibration on the selected channel. The selected channel must be an optical channel.

Example

```
10   DIM Prompt $[64]
20   OUTPUT 707;":CALIBRATE:MODULE:OPTICAL CHAN1"
30   OUTPUT 707;":CALIBRATE:SDONE?"
40   ENTER 707;Prompt$ <Disconnect optical source form channel 1>
50   OUTPUT 707;":CALIBRATE:CONTINUE"
60   OUTPUT 707;":CALIBRATE:SDONE?"
70   ENTER 707;Prompt$ <Enter wavelength and power of optical source>
80   OUTPUT 707;":CALIBRATE:MODULE:OWAVELENGTH 1340E–9"
90   OUTPUT 707;":CALIBRATE:MODULE:OPOWER 500E–6"
100  OUTPUT 707;":CALIBRATE:CONTINUE"
110  OUTPUT 707;":CALIBRATE:SPOWer?"
120  ENTER 707;Prompt$ <Connect optical source to channel 1>
130  OUTPUT 707;":CALIBRATE:CONTINUE"
140  OUTPUT 707;":CALIBRATE:SDONE?"
150  ENTER 707;Prompt$ <Done>
160  END
```

MODule:OWAVelength

Command

:CALibrate:MODule:OWAVelength <wavelength>

This command sets the optical wavelength for an optical channel calibration. This command should only be used for modules with an optical channel.

Example

```
10 OUTPUT 707;":CALIBRATE:MODULE:OWAVELENGTH 1340E–9"
20 END
```

MODule:STATus?

Query

:CALibrate:MODule:STATus? {LMODule | RMODule}

This query returns the module calibration status as either `CALIBRATED` or `UNCALIBRATED`. It will return `UNKNOWN` if the module does not have calibration capability.

Returned Format

[:CALibrate:MODule:STATus] {CALIBRATED | UNCALIBRATED | UNKNOWN} <NL>

MODule:TIME?

Query :CALibrate:MODule:TIME? {LMODule | RMODule}

This query returns the date, time and temperature at which the module was last calibrated. If no module is present in the selected slot, the message “Empty Slot” is returned.

Returned Format [:CALibrate:MODule:TIME] <value><NL>

<value> Is in the format: DD MMM YY HH:MM <delta_temp>

<delta_temp> Is the difference between the current temperature and the temperature when the last calibration was done. For example, <delta_temp> might be:

```
–5C
10C
–12C
```

MODule:VERTical

Command :CALibrate:MODule:VERTical {LMODule | RMODule}

This command initiates a vertical calibration on a selected slot. The specified slot should be the first slot of a double-wide module.

Example GPIB sequence for vertical calibration:

```
10 OUTPUT 707;":CALIBRATE:MODULE:VERTICAL LMODULE" <disconnect all inputs>
20 OUTPUT 707;":CALIBRATE:MODULE:CONTINUE"
30 END
```

OUTPut

Command :CALibrate:OUTPut <dc_value>

This command sets the dc level of the calibrator signal output through the front-panel CAL connector.

Example This example puts a dc voltage of 2.0 V on the analyzer Aux Out connector.

```
10 OUTPUT 707;":CALIBRATE:OUTPUT DC,2.0V"
20 END
```

<dc_value> dc level value in volts, adjustable from –2.5 V to +2.5 Vdc.

Query :CALibrate:OUTPut?

The query returns the current dc level of the calibrator output.

Returned Format [:CALibrate:OUTPut] <dc_value><NL>

Example This example places the current selection for the dc calibration to be printed in the string variable, Selection$, then prints the contents of the variable to the controller's screen.

```
10 DIM Selection$[50]                  !Dimension variable
20 OUTPUT 707;":CALIBRATE:OUTPUT?"
30 ENTER 707;Selection$
40 PRINT Selection$
50 END
```

PROBe:CHANnel<N>

Command :CALibrate:PROBe:CHANnel<N>

This command starts the probe calibration for the selected channel. It has the same action as the command :CHANnel:PROBe:CALibrate.

Example The following example starts calibration for Channel 1.

```
10 OUTPUT 707;":CALibrate:PROBe:CHANnel1"
20 END
```

SAMPlers

Command :CALibrate:SAMPlers {DISable | ENABle}

This command enables or disables the samplers in the module.

Example The following example enables sampler calibration for the module.

```
10 OUTPUT 707;":CALIBRATE:SAMPLERS ENABLE"
20 END
```

Query :CALibrate:SAMPlers?

The query returns the current calibration enable/disable setting.

Returned Format [:CALibrate:SAMPlers]{DISable | ENABle}<NL>

Example The following example gets the current setting for sampler calibration, stores it in the variable Sampler$, and prints the contents of the variable to the controller's screen.

```
10 DIM Sampler$[50]                    !Dimension variable
20 OUTPUT 707;":CALIBRATE:SAMPLERS?"
30 ENTER 707;Sampler$
40 PRINT Sampler$
50 END
```

SDONe?

Query :CALibrate:SDONe?

The CALibrate:SDONe (Step DONe) query will return when the current calibration step is complete.

The contents of the string returned indicates to the user the next step.

Returned Format [:CALibrate:SDONe] <string><NL>

Example This example places the current selection for the calibration pass/fail status to be printed in the string variable, Selection$, then prints the contents of the variable to the controller's screen.

```
10 DIM Selection$[80]                    !Dimension variable
20 OUTPUT 707;":CALIBRATE:SDONE?"
30 ENTER 707;Selection$
40 PRINT Selection$
50 END
```

SKEW

Command :CALibrate:SKEW {CHANnel<N>},<skew_value>

This command sets the channel-to-channel skew factor for a channel. The numerical argument is a real number in seconds which is added to the current time base position to shift the position of the channel's data in time. Use this command to compensate for differences in the electrical lengths of input paths due to cabling and probes.

<N> An integer, from 1 to 4.

<skew_value> A real number.

Example This example sets the analyzer channel 1 skew to 0.1 s.

```
10 OUTPUT 707;":CALIBRATE:SKEW CHANNEL1,0.1s "
20 END
```

Query :CALibrate:SKEW? {CHANnel<N> | EXTernal}

The query returns the current skew value.

Returned Format [:CALibrate:SKEW] <skew_value><NL>

STATus?

Query

:CALibrate:STATus?

This query returns the calibration status of the analyzer. These are nine comma-separated integers, with 1 or 0. A "1" indicates calibrated; a "0" indicates uncalibrated. This matches the status in the Calibration dialog box in the Utilities menu.

Returned Format

[:CALibrate:STATus] <status><NL>

<status>

<Frame Status>,
<Channel1 Vertical>, 0,
<Channel2 Vertical>, 0,
<Channel3 Vertical>,0,
<Channel4 Vertical>, 0

The values that always return “0” are used to make the returned format compatible with the Agilent 83480A and 54750A.

Calibration Procedure

This is an example of how to do module vertical calibration.

```
10    DIM Prompt$[64]
20    OUTPUT 707;":CALIBRATE:MODULE:VERTICAL LMODULE”
30    OUTPUT 707;":CALIBRATE:SDONE?”
40    ENTER 707;Prompt$ <Disconnect everything from left module>
50    OUTPUT 707;":CALIBRATE:CONTINUE”
60    OUTPUT 707;":CALIBRATE:SDONE?”
70    ENTER 707;Prompt$ <Done>
```

Channel Commands

Channel Commands

The CHANnel subsystem commands control all vertical (Y axis) functions of the analyzer. You may toggle the channel displays on and off with the root level commands VIEW and BLANk, or with DISPlay.

BANDwidth

Command

:CHANnel<N>:BANDwidth {HIGH | LOW}

This command controls the channel bandwidth setting. When HIGH, the bandwidth is set to the upper bandwidth limit. When LOW, a lower bandwidth setting is selected in order to minimize broadband noise. See the module manual for cutoff frequency specifications.

<N>

The channel number which represents an integer, 1 to 4. The integer is the slot in which the channel resides.

Example

The following example sets the channel 1 bandwidth to “HIGH”.

```
10 OUTPUT 707;":CHANNEL1:BANDwidth HIGH"
20 END
```

Query

:CHANnel<N>:BANDwidth?

The query returns the state of the bandwidth for the specified channel.

Returned Format

[:CHANnel<N>:BANDwidth] {HIGH | LOW}<NL>

Example

The following example places the current setting of the channel bandwidth in the string variable, Band$, and then prints the contents of the variable to the controller’s screen.

```
10 DIM Limit$[50]                          !Dimension variable
20 OUTPUT 707;":CHANNEL1:BANDwidth?"
30 ENTER 707;Band$
40 PRINT Band$
50 END
```

DISPlay

Command

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

This command turns the display of the specified channel on or off.

<N>

The channel number is an integer 1 to 4.

Example

This example sets channel 1 display to on.

```
10 OUTPUT 707;"CHANNEL1:DISPLAY ON"
20 END
```

Query

:CHANnel<N>:DISPlay?

The query returns the current display condition for the specified channel.

Returned Format

[:CHANnel<N>:DISPlay] {1 | 0}<NL>

Example This example places the current setting of the channel 1 display in the variable Display, then prints the contents of the variable to the controller's screen.

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

FDEScription?

Query :CHANNel<N>:FDEScription?

This query returns the number of filters and a brief description of each filter for channels with more than one internal low-pass filter.

The filter description is the same as the softkey label for the control used to select the active filter.

<N> The channel number is an integer from 1 to 4. The integer is the slot in which the channel resides.

Returned Format [:CHANNel<N>:FDEScription] <n> <,filter1_description> <,filter2_description> <NL>

<n> number of filters

<filter_description> XXX b/s or XXX b/s:N (depending on the module option)

where: XXX is bit rate of filter; N is filter order

FILTer

Command :CHANNel<N>:FILTer {ON | 1 | OFF | 0}

This command controls an internal low-pass filter, if one is present, in the channel hardware.

<N> The channel number is an integer from 1 to 4. The integer is the slot in which the channel resides.

Example

```
10 OUTPUT 707;":CHANNEL1:FILTER ON"
20 END
```

Query :CHANNel<N>:FILTer?

The query returns the filter setting for the specified channel.

Returned Format [:CHANNel<N>:FILTer] {1 | 0}<NL>

Example The following example places the current setting of the filter in the string variable, Filter$, and then prints the contents of the variable to the controller's screen.

```
10 DIM Filter$[50]                    !Dimension variable
20 OUTPUT 707;":CHANNEL1:FILTER?"
30 ENTER 707;Filter$
40 PRINT Filter$
50 END
```

FSELect

Command :CHANnel<N>:FSELect FILTer<filter_number>

This command selects which filter is controlled by on/off for channels with more than one internal low-pass filter.

To query for a description of the filters, see the CHANnel:FDEScription query.

<N> The channel number is an integer from 1 to 4. The integer is the slot in which the channel resides.

<filter_number> The filter number is an integer, 1 or 2. Filter 1 is the top filter on the softkey label.

Example

```
10 OUTPUT 707;":CHANNEL1:FSELECT FILTER1"
20 END
```

Query :CHANnel<N>:FSELect?

The query returns the current filter number for the specified channel.

Returned Format [:CHANnel<N>:FSELect] {FILTer<filter_number>}<NL>

Example The following example places the current setting of the filter in the string variable, Filter$, and then prints the contents of the variable to the controller's screen.

```
10 DIM Filter$[50]                    !Dimension variable
20 OUTPUT 707;":CHANNEL1:FSELECT?"
30 ENTER 707;Filter$
40 PRINT Filter$
50 END
```

See Also CHANnel:FDEScription?

OFFSet

Command :CHANnel<N>:OFFSet <offset_value>

This command sets the voltage that is represented at the center of the display for the selected channel. Offset parameters are probe and vertical scale dependent.

For TDR and TDT applications, when the TDR stimulus is set to differential or common mode, or when OHM, REFLect, or GAIN units are selected, the instrument will change offset to magnify offset. This command is used to set the magnify offset as well as the offset.

<N> The channel number is an integer from 1 to 4, indicating the slot in which the channel resides.

<offset _value> Offset value at center screen. Usually expressed in volts, but could be in other measurement units, such as amperes, if you have specified other units using the CHANnel:UNITs command.

Example This example sets the offset for channel 1 to 0.125 in the current measurement units.

```
10 OUTPUT 707;":CHANNEL1:OFFSET 125E-3"
20 END
```

Query :CHANnel<N>:OFFSet?

The query returns the current offset value for the specified channel.

Returned Format [CHANnel<N>:OFFSet] <offset value><NL>

Example This example places the offset value of the specified channel in the string variable, Offset$, then prints the contents of the variable to the computer's screen.

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

PROBe

Command CHANnel<N>:PROBe: <attenuation factor>[,{RATio | DECibel}]

This command sets the attenuation factor and units. The default attenuation factor is 1:1 and the default units are ratio.

Query CHANnel<N>:PROBe?

When the TDR stimulus is set to differential or common mode, or when OHM, REFLect, or GAIN units are selected, the instrument will change offset to magnify offset. This command is used to set the magnify offset as well as the offset.

Returned Format [:CHANnel<N>:PROBe] <attenuation factor>, {RATio | DECibel}<NL>

PROBe:CALibrate

Command :CHANnel<N>:PROBe:CALibrate

This command starts the probe's calibration for the selected channel. It has the same action as the command :CALibrate:PROBe CHANnel<N> **.**

Example The following example starts calibration for Channel 1.

```
10 OUTPUT 707;":CHANNEL1:PROBE:CALIBRATE"
20 END
```

RANGe

Command :CHANnel<N>:RANGe <range_value>

This command defines the full-scale vertical axis of the selected channel. It sets up acquisition and display hardware to display the waveform at a given range scale. The values represent the full-scale deflection factor of the vertical axis in volts. These values change as the probe attenuation factor is changed.

For TDR and TDT applications, when the TDR stimulus is set to differential or common mode, or when OHM, REFLect, or GAIN units are selected, the instrument will change scale to magnify scale. This command is used to set the magnify range as well as the range.

<N> The channel number is an integer from 1 to 4 indicating the slot in which the channel resides.

<range_value> Full-scale voltage of the specified channel number.

Example This example sets the full-scale range for channel 1 to 500 mV.

```
10 OUTPUT 707;":CHANNEL1:RANGE 500E-3"
20 END
```

Query :CHANnel<N>:RANGe?

The query returns the current full-scale vertical axis setting for the selected channel.

Returned Format [:CHANnel<N>:RANGe]<range value><NL>

Example This example places the current range value in the number variable, Setting, then prints the contents of the variable to the computer's screen.

```
10 OUTPUT 707;":SYSTEM:HEADER OFF”     !Response headers off
20 OUTPUT 707;":CHANNEL1:RANGE?"
30 ENTER 707;Setting
40 PRINT Setting
50 END
```

SCALe

Command :CHANnel<N>:SCALe <scale_value>

This command sets the vertical scale, or units per division, of the selected channel. This command is the same as the front-panel channel scale.

For TDR and TDT applications, when the TDR stimulus is set to differential or common mode, or when OHM, REFLect, or GAIN units are selected, the instrument will change scale to magnify scale. This command is used to set the magnify scale as well as the scale.

<N> The channel number is an integer from 1 to 4.

<scale_value> Vertical scale of the channel in units per division.

Example This example sets the scale value for channel 1 to 500 mV.

```
10 OUTPUT 707;":CHANNEL1:SCALE 500E-3"
20 END
```

Query :CHANnel<N>:SCALe?

The query returns the current scale setting for the specified channel.

Returned Format [:CHANnel<N>:SCALe] <scale value><NL>

Example This example places the current scale value in the number variable, Setting, then prints the contents of the variable to the controller’s screen.

```
10 OUTPUT 707;":SYSTEM:HEADER OFF"     !Response headers off
20 OUTPUT 707;":CHANNEL1:SCALE?"
30 ENTER 707;Setting
40 PRINT Setting
50 END
```

TDRSkew

Command :CHANnel<N>:TDRSkew <percent> [%]

This command sets the TDR skew for the given channel. The TDR skew control moves the TDR step relative to the trigger position. The control may be set from –100 to 100 percent of the allowable range. This command is only applicable to TDR channels.

<N> An integer, 1 through 4, indicating the slot in which the channel resides, followed by an optional A or B identifying which of two possible channels in the slot is being referenced.

<percent> A number between –100 and 100, used to set the step position.

Example The following example sets the TDR skew for channel 1 to 20%.

```
10 OUTPUT 707;":CHANNEL1:TDRSKEW 20"
20 END
```

Query :CHANnel<N>:TDRSkew?

The query returns the current TDR skew setting for the specified channel.It returns the TDR skew value in percent of allowable range from –100 to 100 percent. This command is only applicable to TDR channels. The returned format is a real number.

Returned Format [:CHANnel<N>:TDRSkew] <value><NL>

UNITs

Command :CHANnel<N>:UNITs {VOLT |AMPere | WATT | UNKNown}

This command sets the transducer units.

Query :CHANnel<N>:UNITs?

Returned Format [:CHANnel<N>:UNITs] {VOLT | AMPere | WATT | UNKNown}<NL>

UNITs:ATTenuation

Command :CHANnel<N>:UNITs:ATTenuation <attenuation>

This command sets the transducer attenuation factor.

Query :CHANnel<N>:UNITs:ATTenuation?

Returned Format [:CHANnel<N>:UNITs:ATTenuation] <attenuation><NL>

UNITs:OFFSet

Command :CHANnel<N>:UNITs:OFFSet <offset>

This command sets the transducer offset.

Query :CHANnel<N>:UNITs:OFFSet?

Returned Format [:CHANnel<N>:UNITs:OFFSet] <offset><NL>

WAVelength

Command :CHANnelN:WAVelength {WAVelength1 | WAVelength2 | USER}

This command sets the wavelength selection for optical channels. The Agilent 86101A module will have one factory calibration; all other optical modules will have two. Invoke these calibrations using WAV1 or WAV2. One user-defined wavelength may also be defined via the Channel Calibrate menu. The USER selection is only valid if this user-defined calibration has been performed. The calibration will request the wavelength that the USER choice corresponds to.

This command will also recognize W1310 as an equivalent for WAVelength1 and W1550 for WAVelength2, for compatibility with the Agilent 83480A/ 54750A.

Query :CHANnelN:WAVelength?

The query returns the currently selected wavelength for the channel.

Returned Format [:CHANnelN:WAVelength] {WAV1 | WAV2 | USER}<NL>

Example

```
10 OUTPUT 707;":SYSTEM:HEADER OFF”      !Response headers off
20 OUTPUT 707;":CHANnel1:WAVELENGTH?"
30 ENTER 707;Setting
40 PRINT Setting
50 END UNITs
```

Clock Recovery Commands

Clock Recovery Commands

The Clock RECovery (CREC) subsystem commands control the clock recovery modules. This includes setting data rates, as well as querying locked status and signal present conditions.

LOCKed?

Query

:CRECovery{1 | 3}:LOCKed?

The query returns the locked status of the clock recovery module.

Locked status returns 1, unlocked status returns 0. When a clock rate is selected, unlocked status indicates clock recovery cannot be established and trigger output to the mainframe is disabled. In bypass mode (TOD) locked status is always 0 (unlocked) and triggering is not disabled.

Returned Format

[:CRECovery{1 | 3}:LOCKed] {1 | 0}<NL>

Example

The following example checks the locked status of module in the left slot and places the result in the string variable, Locked$. Then the program prints the contents of the variable to the controller's screen.

```
10 DIM Locked$[50]
20 OUTPUT 707;":CRECOVERY1:LOCKED?"
30 ENTER 707;Locked$
40 PRINT Locked$
50 END
```

RATE

Command

:CRECovery{1 |3}:RATE {TOData | R155 | R622 | R2488 | R1062 | R2125 | R1250 | R2500}

This command sets the clock recovery module data rate based on module slot position: left slot (1), right slot (3). The rates are: Trigger On Data (TOData), Rate 155, Rate 622, Rate 2488, Rate 1062, Rate 2125, Rate 1250, and Rate 2500 in Mb/s.

Rate parameters are nominal and reflect front panel labels and not actual data rates.

Note

After setting a rate, locked status or trigger status should be verified before executing any signal dependent GPIB commands, such as autoscale, or any measurements. This is required to allow the module/instrument enough time to establish a trigger. This can be achieved by querying locked status until locked or generating an event on the module lock.

As noted in the table below, not all modules support the same rates.

Data Rates vs. Model

Rate Parameter	Rate (Mb/s)	Module Model Number		
		83491	**83492**	**83493**
TOData	—	X	X	X
R155	155.52	X	X	X
R622	622.08	X	X	X
R2488	2488.32	X	X	X
R1062	1062.50	X	X	
R2125	2125.00	X	X	
R1250	1250.00	X	X	X
R2500	2500.00	X	X	X

Example This example sets the module in the right slot to a data rate of 2488 Mb/s.

```
10 OUTPUT 707;":CRECOVERY3:RATE R2488"
20 END
```

Query :CRECovery{1 | 3}:RATE?

This query returns the current data rate of the clock recovery module in the specified module position.

Returned Format [:CRECovery{1 | 3}:RATE] {TOData | R155 | R622 | R2488 | R1062 | R2125 | R1250 | R2500}<NL>

Example The following example checks the current data rate of the module in the left slot and places the result in the string variable, Rate$. Then the program prints the contents of the variable to the controller's screen.

```
10 DIM Rate$[50]
20 OUTPUT 707;":CRECOVERY1:RATE?"
30 ENTER 707;Rate$
40 PRINT Rate$
50 END
```

SPResent?

Query

:CRECovery{1 | 3}:SPResent? {RECeiver1 | RECeiver2}

This query returns the status of whether the specified receiver detects an optical signal (Signal PResent). RECeiver2 is used for long wavelengths and RECeiver1 is used for short wavelengths. For electrical clock recovery modules, 83491A, the signal present flags will always return false.

Returned Format

[:CRECovery{1 | 3}:SPResent] {RECeiver1 | RECeiver2}, {1 | 0}<NL>

Example

The following example checks if there is a signal present on receiver two of the module in the right slot and places the result in the string variable, Signal2$. Then the program prints the contents of the variable to the controller's screen.

```
10 DIM Signal2$[50]
20 OUTPUT 707;":CRECOVERY3:SPRESENT? RECEIVER2"
30 ENTER 707;Signal2$
40 PRINT Signal2$
50 END
```

14

Disk Commands

Disk Commands

The DISK subsystem commands perform the disk operations as defined in the Disk menu. This allows storage and retrieval of waveforms and setups, as well as formatting the disk.

Some commands in this subsystem operate only on files and directories on “A:\”, under “C:\User Files”, or on any mapped network drive, and are noted in the command section.

Enclose File Name in Quotation Marks

When specifying a file name, you must enclose it in quotation marks.

CDIRectory

This command operates only on files and directories on “A:\”, under “C:\User Files”, or on any mapped network drive.

Command :DISK:CDIRectory ["<directory>" | {CGRade | LSUMmaries | ROOT | SETups | SIMages | SMASks | TDRTDT | UMASks | WAVeforms}]

This command changes the present working directory to the designated directory name. If an error occurs, the requested directory does not exist. You can view the error with the :SYSTem:ERRor? [{NUMBer | STRing}] query.

<directory> A character-quoted ASCII string, which can include the subdirectory designation. You must separate the directory name and any subdirectories with a backslash (\).

ROOT This parameter changes the working directory to “C:\User Files”.

Example

```
10 OUTPUT 707;":DISK:CDIRECTORY ""C:\USER FILES\DATA"""
20 END
```

CDIR "C:\" Is Not Allowed

You can execute the command CDIR "A:\", but the command CDIR "C:\" is not allowed. If you attempt to execute CDIR "C:\", the present working directory (PWD) is not changed. The directory specified *must* be below “C:\User Files\”.

DELete

This command operates only on files and directories on “A:\”, under “C:\User Files”, or on any mapped network drive.

Command :DISK:DELete "<file_name>"

This command deletes a file from the disk. An error is displayed on the analyzer screen if the requested file does not exist. The file “C:\User Files” *cannot* be deleted.

<file_name> A character-quoted ASCII string which can include subdirectories with the name of the file.

Example

```
10 OUTPUT 707;":DISK:DELETE ""FILE1.SET"""
20 END
```

DIRectory?

Query :DISK:DIRectory? ["<directory>" | {CGRade | ROOT | LSUMmaries | SETups | SIMages | SMASks | TDRTDT | UMASks | WAVeforms}]

This query returns the requested directory listing. The directory may be specified as a string, such as "C:\User Files\waveforms", or as a parameter. If no parameter is used, a listing of the present working directory is returned.

<directory> The list of file names and directories.

Returned Format [:DISK:DIRectory]<n><NL><directory><NL>

<n> The specifier that is returned before the directory listing, indicating the number of lines in the listing.

<directory> The list of filenames and directories. Each line is separated by a <NL>.

Example This example displays a number, then displays a list of files and directories in the current directory. The number indicates the number of lines in the listing.

```
10 DIM A$[80]
20 INTEGER Num_of_lines
30 OUTPUT 707;":DISK:DIR?"
40 ENTER 707;Num_of_lines
50 PRINT Num_of_lines
60 FOR I=1 TO Num_of_lines
70 ENTER 707;A$
80 PRINT A$
90 NEXT I
100 END
```

LOAD

This command operates only on files and directories on “A:\”, under “C:\User Files”, or on any mapped network drive.

Command :DISK:LOAD "<file_name>"[,<destination>]

This command restores a setup, waveform, or TDR/TDT calibration from the disk. The type of file is determined by the filename suffix if one is present, or by the destination field if one is not present. If a destination is specified, it takes precedence over the filename suffix. You can load .wfm, .txt, .cgs, .msk,

.set and .tdr file types. The TDRTDT option is a file type choice used to load TDR/TDT calibration values into the instrument. For more information on loading files, see "File Names and Types" on page 1-11, and "File Locations" on page 1-13.

<file_name> The filename, with a 3-character extension. You can use either .wfm, .txt, .cgs, .msk, .set, or .tdr as a suffix after the filename. If no file suffix is specified, the default is .wfm.

The default directory for the file type is assumed, or you can specify the entire path. For example, you can load the standard setup file "setup0.set" using the command:

```
:DISK:LOAD "C:\User Files\Setups\setup0.set",setup
```

The default destination for .txt and .wfm files is WMEMory1.

<destination> {CGRade | MASK | WMEMory<N> | SETup | TDRTDT}

<N> An integer from 1 to 4.

Example

```
10 OUTPUT 707;":DISK:LOAD ""FILE1.WFM"",WMEM1"
20 END
```

MDIRectory

> This command operates only on files and directories on "A:\", under "C:\User Files", or on any mapped network drive.

Command :DISK:MDIRectory "<directory>"

This command creates a directory in the present working directory, with the designated directory name. An error is displayed if the requested path does not exist.

<directory> A character-quoted ASCII string which can include subdirectories. You must separate the directory name and any subdirectories with a backslash (\).

Example

```
10 OUTPUT 707;":DISK:MDIRECTORY ""CPROGRAMS"""
20 END
```

PWD?

Query :DISK:PWD?

This query returns the name of the present working directory (including the full path).

Returned Format [:DISK:PWD] <present_working_directory><NL>

Example

```
10 DIM Wdir$[200]
20 OUTPUT 707;":DISK:PWD?"
30 ENTER 707; Wdir$
40 PRINT Wdir$
50 END
```

STORe

This command operates only on files and directories on “A:\”, under “C:\User Files”, or on any mapped network drive.

Command :DISK:STORe <source>,"<file_name>"[,<format>]

This command stores a setup, waveform or TDR response to the disk. The filename does not include a suffix. The suffix is supplied by the analyzer depending on the source and file format specified. The TDRTDT option is a file type choice used to store the instrument’s TDR/TDT calibration values. For more information on storing files, see “File Names and Types” on page 1-11, and “File Locations” on page 1-13.

<source> {CHANnel<N> | FUNCtion<N> | WMEMory<N> | SETup | RESPonse<N> | TDRTDT}

<N> An integer from 1 to 4, representing the channel, function, TDR response or waveform memory number.

<file_name> Name of the file, with a maximum of 254 characters (including the path name, if used). The filename assumes the present working directory if a path does not precede the file name.

<format> {TEXT {,<YVALues> | <VERBose>} | INTernal}

Fields and Default Values

The format field is for waveforms, and the default is INTernal. In TEXT mode, y values may be specified so that only the y values are stored. VERBose is the default in which y values and the waveform preamble are stored. Only waveforms of 128K or less may be written to disk in the TEXT formats. See Chapter 26, “Waveform Commands” for information on converting data to values.

Example

```
10 OUTPUT 707;":DISK:STORE SET,""FILE1"""
20 END
```

Display Commands

Display Commands

The DISPlay subsystem controls the display of data, markers, text, graticules, and the use of color. You select the display mode using the ACQuire:TYPE command. Select the number of averages using ACQuire:COUNt.

CGRade:LEVels?

Query

:DISPlay:CGRade:LEVels?

This query returns the range of hits represented by each color. Fourteen values are returned, representing the minimum and maximum count for each of seven colors. The values are returned in the following order:

- Greatest intensity color minimum
- Greatest intensity color maximum
- Next greatest intensity color minimum
- Next greatest intensity color maximum
- . . . .
- Least intensity color minimum
- Least intensity color maximum

Returned Format

[:DISPlay:CGRade:LEVels] <color format><NL>

<color format>

<intensity color min / max> is an integer value from 0 to 63.488.

Example

The following example gets the range of hits represented by each color and prints it on the controller screen.

```
10 DIM Setting$[50]                              !Dimension variable
20 OUTPUT 707;":DISPLAY:CGRADE:LEVELS?"
30 ENTER 707;Cgrade$
40 PRINT Cgrade$
50 END
```

CONNect

Command

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

When enabled, :DISPlay:CONNect draws a line between consecutive waveform data points. This is also known as linear interpolation. This command has no effect on color grade or gray scale displays.

Example

This example turns on the connect-the-dots feature.

```
10 OUTPUT 707;":DISPLAY:CONNECT ON"
20 END
```

Query

:DISPlay:CONNect?

The query returns the status of the connect-the-dots feature.

Returned Format

[:DISPlay:CONNect] {1 | 0}<NL>

DATA?

Query	:DISPlay:DATA? [<type>[,<screen_mode>[,<compression> [,<inversion>]]]]
	The query returns information about the captured data. If no options to the query are specified, the default selections are PCX file type, SCReen mode, comparison turned ON, and inversion set to NORMal.
<type>	The file type: BMP | PCX | EPS | PS | GIF.
<screen_mode>	The display setting: SCReen | GRATicule.
<compression>	The file compression feature: ON | OFF.
<inversion>	The inversion of the displayed file: NORMal | INVert.
Returned Format	[:DISPlay:DATA] <binary_block_data><NL>
<binary_block_data>	Data in the IEEE 488.2 definite block format.

DCOLor (Default COLor)

Command

```
:DISPlay:DCOLor
```

This command (Default COLor) resets the screen colors to the predefined factory default colors. It also resets the grid intensity.

Example

This example sends the DCOLor command.

```
10 OUTPUT 707;":DISPLAY:DCOLOR"
20 END
```

GRATicule

Commands

:DISPlay:GRATicule {GRID|FRAMe}

:DISPlay:GRATicule:INTensity <intensity_value>

These commands select the type of graticule that is displayed. 86100A analyzers have a 10-by-8 (unit) display graticule grid that you can turn on or off. When the grid is on, a grid line is place on each vertical and horizontal division. When it is off, a frame with tic marks surrounds the graticule edges.

<intensity_value>

A number from 0 to 100, indicating the percentage of display intensity.

You can dim the grid's intensity or turn the grid off to better view waveforms that might be obscured by the graticule lines. Otherwise, you can use the grid to estimate waveform measurements such as amplitude and period.

When printing, the grid intensity control doesn't affect the hardcopy. To remove the grid from a printed hardcopy, you must turn off the grid before printing.

Example

This example sets up the analyzer's display background with a frame that is separated into major and minor divisions.

```
10 OUTPUT 707;":DISPLAY:GRATICULE FRAME"
20 END
```

Queries

:DISPlay:GRATicule?

:DISPlay:GRATicule:INTensity?

The queries return the type of graticule currently displayed, or the intensity, depending on the query you request.

Returned Format

[:DISPlay:GRATicule] {GRID|FRAMe}<NL>

[:DISPlay:GRATicule:INTensity] <value><NL>

Example

This example places the current display graticule setting in the string variable, Setting$, then prints the contents of the variable to the controller's screen.

```
10 DIM Setting$[50]                          !Dimension variable
20 OUTPUT 707;":DISPLAY:GRATICULE?"
30 ENTER 707;Setting$
40 PRINT Setting$
50 END
```

LABel

Command	:DISPlay:LABel "<string_argument>" [,<row>[,<column>[,<text_color>[,<background>]]]]
	This command allows you to place a label on the graticule area of the display. The operator should periodically clear the labels using the LABel:DALL command.
<string_argument>	Any series of ASCII characters enclosed in quotation marks.
<row>	0 to 12, where 0 is the top row and the default
<column>	0 to 61, where 0 is the left column and the default
<text_color>	{CHANnel<N> | WHITe} Default is WHITe
<background>	{OPAQue | TRANsparent} Default is TRANsparent
Example	This example places a label on the upper left corner of the graticule.

```
10 OUTPUT 707;":DISPLAY:LABEL""This is a label"""
20 END
```

LABel:DALL

Command	:DISPlay:LABel:DALL
	This command deletes all labels.
Example	This example deletes all labels.

```
10 OUTPUT 707;":DISPLAY:LABEL:DALL"
20 END
```

PERSistence

Command	:DISPlay:PERSistence {MINimum | INFinite | <persistence_value> | CGRade | GSCale}
	This command sets the display persistence. It works in both real time and equivalent time modes. The parameter for this command can be either MINimum (zero persistence), INFinite, or a real number from 0.1 to 40.0, representing the persistence in seconds.
<persistence_value>	A real number, 0.1 to 40.0, representing the persistence in seconds.
Mode	Eye mode only for CGRade and GSCale arguments.

Example

This example sets the persistence to infinite.

```
10 OUTPUT 707;":DISPLAY:PERSISTENCE INFINITE"
20 END
```

Query

:DISPlay:PERSistence?

The query returns the current persistence value.

Returned Format

[:DISPlay:PERSistence] {MINimum | INFinite | <value> | CGRade | GSCale}<NL>

Example

This example places the current persistence setting in the string variable, Setting$, then prints the contents of the variable to the controller's screen.

```
10 DIM Setting$[50]                       !Dimension variable
20 OUTPUT 707;":DISPLAY:PERSISTENCE?"
30 ENTER 707;Setting$
40 PRINT Setting$
50 END
```

SCOLor

Command

:DISPlay:SCOLor <color_name>, <hue>, <saturation>, <luminosity>

The DISPlay:SCOLor command sets the color of the specified display element and restores the colors to their factory settings. The display elements are described in Table 15-1 on page 15-7.

<color_name>

{CGRade1 | CGRADE2 | CGRADE3 | CGRADE4 | CGRADE5 | CGRADE6 | CGRade7 | CHANnel1 | CHANnel2 | CHANnel3 | CHANnel4 | GRID | MARGin | MARKers | MASK | MEASurements | WBACkgrnd | WOVerlap | WMEMories | WINText | WINBackgrnd}

Table 15-1. Color Names

Color Name	Definition
CGRADE1	First range of pixel counts for the color grade persistence display
CGRADE2	Second range of pixel counts for the color grade persistence display
CGRADE3	Third range of pixel counts for the color grade persistence display
CGRADE4	Fourth range of pixel counts for the color grade persistence display
CGRADE5	Fifth range of pixel counts for the color grade persistence display
CGRADE6	Sixth range of pixel counts for the color grade persistence display
CGRADE7	Seventh range of pixel counts for the color grade persistence display
CHANnel1	Channel 1 waveform display element.
CHANnel2	Channel 2 waveform display element.

Table 15-1. Color Names (Continued)

Color Name	Definition
CHANnel3	Channel 3 waveform display element.
CHANnel4	Channel 4 waveform display element.
GRID	Display element for the grid inside the waveform viewing area.
MARGin	Display element for the margins.
MARKers	Display element for the markers.
MASK	Display element for the masks.
MEASurements	Display element for the measurements text.
WBACkgrnd	Display element for the waveform viewing area's background.
WOVerlap	Display element for waveforms when they overlap each other.
WMEMories	Display element for waveform memories.
WINText	Display element used in dialog box controls and pull-down menus.
WINBackgrnd	Display element for the background color used in dialog boxes and buttons.

<hue> The hue control sets the color of the chosen display element. As hue is increased from 0%, the color changes from red, to yellow, to green, to blue, to purple, then back to red again at 100% hue. For color examples, see the sample color settings table in the 86100A on-line help file. Pure red is 100%, pure blue is 67%, and pure green is 33%.

<saturation> The saturation control sets the color purity of the chosen display element. The saturation of a color is the purity of a color or the absence of white. A 100% saturated color has no white component. A 0% saturated color is pure white.

<luminosity> The luminosity control sets the color brightness of the chosen display element. A 100% luminosity is the maximum color brightness. A 0% luminosity is pure black.

Example This example sets the hue to 50, the saturation to 70, and the luminosity to 90 for the markers.

```
10 OUTPUT 707;":DISPLAY:SCOLOR MARKERS,50,70,90"
20 END
```

Query :DISPlay:SCOLor? <color_name>

The query returns the hue, saturation, and luminosity for the specified color.

Returned Format [:DISPlay:SCOLor] <color_name>, <hue>, <saturation>, <luminosity><NL>

Example

This example places the current settings for the graticule color in the string variable, Setting$, then prints the contents of the variable to the controller's screen.

```
10 DIM Setting$[50]                    !Dimension variable
20 OUTPUT 707;":DISPLAY:SCOLOR? GRID"
30 ENTER 707;Setting$
40 PRINT Setting$
50 END
```

SSAVer

Commands

:DISPlay:SSAVer {DISabled|ENABled}

:DISPlay:SSAVer:AAFTer <time>

These commands let you disable or enable the analyzer screen saver, and specify a time before the screen saver turns on.

<time>

An integer; either 2, 3, 4, 5, 6, 7, or 8. The time value specifies the amount of time, in hours, that must pass before the screen saver will turn on.

Example

This example enables the analyzer screen saver.

```
10 OUTPUT 707;":DISPLAY:SSAVER ENABLED"
20 OUTPUT 707;":DISPLAY:SSAVER:AAFT 4"
30 END
```

Queries

:DISPlay:SSAVer?

:DISPlay:SSAVer:AAFTer?

The queries return the state of the screen saver.

Returned Format

[:DISPlay:SSAVer] {DISabled|ENABled}<NL>

[:DISPlay:SSAVer:AAFTer] <time><NL>

16

Function Commands

Function Commands

The FUNCtion subsystem defines functions 1–4. The operands of these functions can be any of the installed channels in the analyzer, waveform memories 1–4, functions 1–4, or a constant.

The vertical scaling and offset functions can be controlled remotely using the RANGe and OFFSet commands in this subsystem. You can obtain the horizontal scaling and position values of the functions using the HORizontal:RANGe and HORizontal:POSition queries in this subsystem.

If a channel is not on but is used as an operand, then that channel will acquire waveform data.

If the operand waveforms have different memory depths, the function uses the shorter of the two.

If the two operands have the same time scales, the resulting function has the same time scale. If the operands have different time scales, the resulting function has no valid time scale. This is because operations are performed based on the displayed waveform data position, and the time relationship of the data records cannot be considered. When the time scale is not valid, delta time pulse parameter measurements have no meaning, and the unknown result indicator is displayed on the screen.

Constant operands take on the same time scale as the associated waveform operand.

DISPlay

Command :FUNCtion<N>:DISPlay {{ON | 1} | {OFF | 0}}

This command either displays the selected function or removes it from the display.

<N> An integer, 1–4, representing the selected function.

Example This example turns function 1 on.

```
10 OUTPUT 707;":FUNCTION1:DISPLAY ON"
20 END
```

Query :FUNCtion<N>:DISPlay?

The query returns the displayed status of the specified function.

Returned Format [:FUNCtion<N>:DISPlay] {1 | 0}<NL>

Example This example places the current state of function 1 in the variable, Setting, then prints the contents of the variable to the computer's screen.

```
10 OUTPUT 707;":SYSTEM:HEADER OFF"
20 OUTPUT 707;":FUNCTION1:DISPLAY?"
30 ENTER 707;Setting
40 PRINT Setting
50 END
```

FUNCtion<N>?

Query

:FUNCtion<N>?

This query returns the currently defined source(s) for the function.

Returned Format

[:FUNCtion<N>] {<operand>,[,<operand>]}<NL>

<N>

An integer, 1–4, representing the selected function.

<operator>

Active math operation for the selected function: ADD, DIFF, DIVide, FFTMagnitude, INTegrate, INVert, MAGNify, MAXimum, MINimum, MULTiply, SUBTract, or VERSus.

<operand>

Any allowable source for the selected FUNCtion, including channels 1–4, waveform memories 1–4, or functions 1–4. If the function is applied to a constant, the source returns the constant.

Example

This example returns the currently defined source for function 1.

```
10 OUTPUT 707;":FUNCTION1?"
20 END
```

If the headers are off (see :SYSTem:HEADers), the query returns only the operands, not the operator.

```
10 :SYST:HEAD ON
20 :FUNC1:ADD CHAN1,CHAN2
30 :FUNC1? !returns :FUNC1:ADD CHAN1,CHAN2
40 :SYST:HEAD OFF
50 :FUNC1? !returns CHAN1,CHAN2
```

HORizontal

Command

:FUNCtion<N>:HORizontal {AUTO | MANual}

This command sets the horizontal tracking to either AUTO or MANual.

The HORizontal command also includes a subsystem consisting of the following commands and queries, which are described on the following pages:

- POSition
- RANGe

<N>

An integer, 1–4, representing the selected function.

Query

:FUNCtion<N>:HORizontal?

The query returns the current horizontal scaling mode of the specified function.

Returned Format

[:FUNCtion<N>:HORizontal] {AUTO | MANual}<NL>

Example This example places the current state of function 1 horizontal tracking in the string variable, Setting$, then prints the contents of the variable to the computer's screen.

```
10 DIM Setting$[50]                          !Dimension variable
20 OUTPUT 707;":FUNCTION1:HORIZONTAL?"
30 ENTER 707;Setting$
40 PRINT Setting$
50 END
```

HORizontal:POSition

Command :FUNCtion<N>:HORizontal:POSition <position_value>

This command sets the time value at center screen for the selected function.

<N> An integer, 1–4, representing the selected function.

<position_value> Position value in time, in seconds.

Query :FUNCtion<N>:HORizontal:POSition?

The query returns the current time value at center screen of the selected function.

Returned Format [:FUNCtion<N>:HORizontal:POSition] <position><NL>

Example This example places the current horizontal position setting for function 2 in the numeric variable, Value, then prints the contents to the computer's screen.

```
10 OUTPUT 707;":SYSTEM:HEADER OFF"           !Response headers off
20 OUTPUT 707;":FUNCTION2:HORIZONTAL:POSITION?"
30 ENTER 707;Value
40 PRINT Value
50 END
```

HORizontal:RANGe

Command :FUNCtion<N>:HORizontal:RANGe <range_value>

This command sets the current time range for the specified function. This automatically selects manual mode.

<N> An integer, 1–4, representing the selected function.

<range_value> Width of screen in current X-axis units (usually seconds).

Query :FUNCtion<N>:HORizontal:RANGe?

The query returns the current time range setting of the specified function.

Returned Format [:FUNCtion<N>:HORizontal:RANGe] <range><NL>

Example This example places the current horizontal range setting of function 2 in the numeric variable, Value, then prints the contents to the computer's screen.

```
10 OUTPUT 707;":SYSTEM:HEADER OFF"             !Response headers off
20 OUTPUT 707;":FUNCTION2:HORIZONTAL:RANGE?"
30 ENTER 707;Value
40 PRINT Value
50 END
```

INVert

Command :FUNCtion<N>:INVert <operand>

This command defines a function that inverts the defined operand's waveform by multiplying by –1.

<N> An integer, 1–4, representing the selected function.

<operand> {CHANnel<n> | FUNCtion<n> | RESPonse<n> | WMEMory<n> | <float_value>}

<n> An integer from 1 to 4.

Example This example sets up function 2 to invert the signal on channel 1.

```
10 OUTPUT 707;":FUNCTION2:INVERT CHANNEL1"
20 END
```

Functions Used as Operands

A function may be used as a source for another function, subject to the following constraints:

F4 can have F1, F2, or F3 as a source.

F3 can have F1 or F2 as a source.

F2 can have F1 as a source.

F1 cannot have any other function as a source.

MAGNify

Command :FUNCtion<N>:MAGNify <operand>

This command defines a function that is a copy of the operand. The magnify function is a software magnify. No hardware settings are altered as a result of using this function. It is useful for scaling channels, another function, TDR/TDT responses or memories with the RANGe and OFFSet commands in this subsystem.

<N> An integer, 1–4, representing the selected function.

<operand> {CHANnel<n> | FUNCtion<n> | RESPonse<n> | WMEMory<n> | <float_value>}

<n> An integer from 1 to 4.

Example This example creates a function (function 1) that is a magnified version of channel 1.

```
10 OUTPUT 707;":FUNCTION1:MAGNIFY CHANNEL1"
20 END
```

Functions Used as Operands

A function may be used as a source for another function, subject to the following constraints:

F4 can have F1, F2, or F3 as a source.

F3 can have F1 or F2 as a source.

F2 can have F1 as a source.

F1 cannot have any other function as a source.

OFFSet

Command :FUNCtion<N>:OFFSet <offset_value>

This command sets the voltage represented at the center of the screen for the selected function. This automatically changes the mode from auto to manual.

<N> An integer, 1–4, representing the selected function.

<offset_value> The offset value is limited to being within the vertical range that can be represented by the function data.

Example This example sets the offset voltage for function 1 to 2 mV.

```
10 OUTPUT 707;":FUNCTION1:OFFSET 2E-3"
20 END
```

Query :FUNCtion<N>:OFFSet?

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

Returned Format [:FUNCtion<N>:OFFSet] <offset_value><NL>

Example This example places the current setting for offset on function 2 in the numeric variable, Value, then prints the result to the computer's screen.

```
10 OUTPUT 707;":SYSTEM:HEADER OFF"      !Response headers off
20 OUTPUT 707;":FUNCTION2:OFFSET?"
30 ENTER 707;Value
40 PRINT Value
50 END
```

RANGe

Command :FUNCtion<N>:RANGe <full_scale_range>

This command defines the full scale vertical axis of the selected function. This automatically changes the mode from auto to manual.

<N> An integer, 1–4, representing the selected function.

<full_scale_range> The full-scale vertical range.

Example This example sets the full scale range for function 1 to 400 mV.

```
10 OUTPUT 707;":FUNCTION1:RANGE 400E-3"
20 END
```

Query :FUNCtion<N>:RANGe?

The query returns the current full scale range setting for the specified function.

Returned Format [:FUNCtion<N>:RANGe] <full_scale_range><NL>

Example This example places the current range setting for function 2 in the numeric variable “Value,” then prints the contents to the computer screen.

```
10 OUTPUT 707;":SYSTEM:HEADER OFF"      !Response headers off
20 OUTPUT 707;":FUNCTION2:RANGE?"
30 ENTER 707;Value
40 PRINT Value
50 END
```

SUBTract

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

This command defines a function that algebraically subtracts the second operand from the first operand.

<N> An integer, 1–4, representing the selected function.

<operand> {CHANnel<n> | FUNCtion<n> | RESPonse<n> | WMEMory<n> | <float_value>}

<n> An integer from 1 to 4.

Example This example defines a function that subtracts waveform memory 1 from channel 1.

```
10 OUTPUT 707;":FUNCTION1:SUBTRACT CHANNEL1,WMEMORY1"
20 END
```

Functions Used as Operands

A function may be used as a source for another function, subject to the following constraints:

F4 can have F1, F2, or F3 as a source.

F3 can have F1 or F2 as a source.

F2 can have F1 as a source.

F1 cannot have any other function as a source.

VERSus

Command :FUNCtion<N>:VERSus <operand>,<operand>

This command defines a function for an X-versus-Y display. The first operand defines the Y axis and the second defines the X axis. The Y-axis range and offset are initially equal to that of the first operand and can be adjusted with the RANGe and OFFSet commands in this subsystem.

<N> An integer, 1–4, representing the selected function.

<operand> {CHANnel<n> | FUNCtion<n> | RESPonse<n> | WMEMory<n> | <float_value>}

<n> An integer from 1 to 4.

Example

This example defines function 1 as an X-versus-Y display. Channel 1 is the X axis and waveform memory 2 is the Y axis.

```
10 OUTPUT 707;":FUNCTION1:VERSUS WMEMORY2,CHANNEL1"
20 END
```

Functions Used as Operands

A function may be used as a source for another function, subject to the following constraints:

F4 can have F1, F2, or F3 as a source.

F3 can have F1 or F2 as a source.

F2 can have F1 as a source.

F1 cannot have any other function as a source.

VERTical

Command

:FUNCtion<N>:VERTical {AUTO | MANual}

This command sets the vertical scaling mode of the specified function to either AUTO or MANual.

The VERTical command also contains a subsystem consisting of the following commands and queries:

- OFFset
- RANge

<N>

An integer, 1–4, representing the selected function.

Query

:FUNCtion<N>:VERTical?

The query returns the current vertical scaling mode of the specified function.

Returned Format

[:FUNCtion<N>:VERTical] {AUTO | MANual}<NL>

Example

This example places the current state of the vertical tracking of function 1 in the string variable, Setting$, then prints the contents of the variable to the computer's screen.

```
10 DIM Setting$[50]                  !Dimension variable
20 OUTPUT 707;":FUNCTION1:VERTICAL?"
30 ENTER 707;Setting$
40 PRINT Setting$
50 END
```

VERTical:OFFSet

Command :FUNCtion<N>:VERTical:OFFSet <offset_value>

This command sets the voltage represented at center screen for the selected function. This automatically changes the mode from auto to manual.

<N> An integer, 1–4, representing the selected function.

<offset_value> The offset value is limited only to being within the vertical range that can be represented by the function data.

Query :FUNCtion<N>:VERTical:OFFset?

The query returns the current offset value of the selected function.

Returned Format [:FUNCtion<N>:VERTical:OFFset] <offset_value><NL>

Example This example places the current offset setting for function 2 in the numeric variable, Value, then prints the contents to the computer's screen.

```
10 OUTPUT 707;":SYSTEM:HEADER OFF"              !Response headers off
20 OUTPUT 707;":FUNCTION2:VERTICAL:OFFSET?"
30 ENTER 707;Value
40 PRINT Value
50 END
```

VERTical:RANGe

Command :FUNCtion<N>:VERTical:RANGe <full_scale_range>

This command defines the full-scale vertical axis of the selected function. This automatically changes the mode from auto to manual, if the scope is not already in manual mode.

<N> An integer, 1–4, representing the selected function.

<full_scale_range> The full-scale vertical range.

Query :FUNCtion<N>:VERTical:RANGe?

The query returns the current range setting of the specified function.

Returned Format [:FUNCtion<N>:VERTical:RANGe] <range><NL>

Example

This example places the current vertical range setting of function 2 in the numeric variable, Value, then prints the contents to the computer screen.

```
10 OUTPUT 707;":SYSTEM:HEADER OFF"          !Response headers off
20 OUTPUT 707;":FUNCTION2:VERTICAL:RANGE?"
30 ENTER 707;Value
40 PRINT Value
50 END
```

Hardcopy Commands

Hardcopy Commands

The HARDcopy subsystem commands set various parameters for printing the screen. The print sequence is activated when the root level :PRINt command is sent.

AREA

Command :HARDcopy:AREA {GRATicule | SCReen}

This command selects which data from the screen is to be printed. When you select GRATicule, only the graticule area of the screen is printed (this is the same as choosing Waveforms Only in the Configure Printer dialog box). When you select SCReen, the entire screen is printed.

Example This example selects the graticule for printing.

```
10 OUTPUT 707;":HARDCOPY:AREA GRATICULE"
20 END
```

Query :HARDcopy:AREA?

The query returns the current setting for the area of the screen to be printed.

Returned Format [:HARDcopy:AREA] {GRATicule | SCReen}<NL>

Example This example places the current selection for the area to be printed in the string variable, Selection$, then prints the contents of the variable to the computer's screen.

```
10 DIM Selection$[50]              !Dimension variable
20 OUTPUT 707;":HARDCOPY:AREA?"
30 ENTER 707;Selection$
40 PRINT Selection$
50 END
```

DPRinter

Command :HARDcopy:DPRinter {<printer_number>|<printer_string>}

This command selects the default printer to be used.

<printer_number> An integer representing the attached printer. This number corresponds to the number returned with each printer name by the ":HARDcopy:PRINters?" query.

<printer_string> A string of alphanumeric characters representing the attached printer.

The HARDcopy:DPRinter command specifies a number or string for the printer attached to the analyzer. The printer_string must exactly match the character strings in the File, Print Setup dialog boxes, or the strings returned by the ":HARDcopy:PRINters?" query.

Examples This example sets the default printer to the second installed printer returned by the :HARDcopy:PRINters? query.

```
10 OUTPUT 707;":HARDCOPY:DPRINTER 2"
20 END
```

This example sets the default printer to the installed printer with the name "HP Laser".

```
10 OUTPUT 707;":HARDCOPY:DPRINTER ""HP Laser"""
20 END
```

Query :HARDcopy:DPRinter?

The query returns the current printer number and string.

Returned Format [:HARDcopy:DPRinter?] {<printer_number>,<printer_string>,DEFAULT}<NL>

Or, if there is no default printer (no printers are installed), only a <NL> is returned.

Example This example places the current setting for the hardcopy printer in the string variable, Setting$, then prints the contents of the variable to the computer's screen.

```
10 DIM Setting$[50]                        !Dimension variable
20 OUTPUT 707;":HARDCOPY:DPRinter?"
30 ENTER 707;Setting$
40 PRINT Setting$
50 END
```

Programs Must Wait After Changing the Default Printer

It takes several seconds to change the default printer. Any programs that try to set the default printer must wait (10 seconds is a safe amount of time) for the change to complete before sending other commands. Otherwise the analyzer will become unresponsive.

FACTors

Command :HARDcopy:FACTors {{ON | 1}|{OFF | 0}}

This command determines whether the analyzer setup factors will be appended to screen or graticule images. FACTors ON is the same as choosing Include Setup Information in the Configure Printer dialog box.

Example This example turns on the setup factors.

```
10 OUTPUT 707;":HARDCOPY:FACTORS ON"
20 END
```

Query :HARDcopy:FACTors?

The query returns the current setup factors setting.

Returned Format [:HARDcopy:FACTors] {1|0}<NL>

Example This example places the current setting for the setup factors in the string variable, Setting$, then prints the contents of the variable to the computer's screen.

```
10 DIM Setting$[50]                    !Dimension variable
20 OUTPUT 707;":HARDCOPY:FACTORS?"
30 ENTER 707;Setting$
40 PRINT Setting$
50 END
```

IMAGe

Command :HARDcopy:IMAGe {NORMal | INVert | MONochrome}

This command prints the image normally, inverted, or in monochrome. IMAGe INVert is the same as choosing Invert Waveform Colors in the Configure Printer dialog box.

Example This example sets the hardcopy image output to normal.

```
10 OUTPUT 707;":HARDCOPY:IMAGE NORMAL"
20 END
```

Query :HARDcopy:IMAGe?

The query returns the current image setting.

Returned Format [:HARDcopy:IMAGe] {NORMal | INVert | MONochrome}<NL>

Example This example places the current setting for the hardcopy image in the string variable, Setting$, then prints the contents of the variable to the computer's screen.

```
10 DIM Setting$[50]                    !Dimension variable
20 OUTPUT 707;":HARDCOPY:IMAGE?"
30 ENTER 707;Setting$
40 PRINT Setting$
50 END
```

PRINters?

Query :HARDcopy:PRINters?

This query returns the currently available printers.

Returned Format [:HARDcopy:PRINters]<printer_count><NL><printer_data><NL>[,<printer_data><NL>]

<printer_count> Number of printers currently installed.

<printer_data> The printer number and the name of an installed printer. The word DEFAULT appears next to the printer that is the currently selected default printer.

Example This example places the number of installed printers into the variable Count, loops through that number of times, and prints the installed printer names to the computer screen.

```
10 DIM Setting$[50]                          !Dimension variable
20 OUTPUT 707;":HARDCOPY:PRINTERS?"
30 ENTER 707;Count
40 IF Count>0 THEN
50 FOR Printer_number=1 TO Count
60 ENTER 707;Setting$
70 PRINT Setting$
80 NEXT Printer_number
90 END IF
100 END
```

18

Histogram Commands

Histogram Commands

The Histogram commands and queries control the histogram features. A histogram is a probability distribution that shows the distribution of acquired data within a user-definable histogram window. You can display the histogram either vertically, for voltage measurements, or horizontally, for timing measurements.

The most common use for histograms is measuring and characterizing noise or jitter on displayed waveforms. Noise is measured by sizing the histogram window to a narrow portion of time and observing a vertical histogram that measures the noise on a waveform. Jitter is measured by sizing the histogram window to a narrow portion of voltage and observing a horizontal histogram that measures the jitter on an edge.

Histograms and the Database

The histograms, mask testing, and color-graded (including gray scale) display use a specific database that uses a different memory area from the waveform record for each channel. When any of these features are turned on, the instrument starts building the database. The database is the size of the graticule area. Behind each pixel is a 16-bit counter that is incremented each time data from a channel or function hits a pixel. The maximum count (saturation) for each counter is 63,488. You can use the :MEASure:CGRade:PEAK? or DISPlay:CGRade:LEVels? queries to see if any of the counters are close to saturation.

The database continues to build until the instrument stops acquiring data or all three functions (color-graded display, mask testing, and histograms) are turned off. You can set the ACQuisition:RUNTil (Run Until) mode to stop acquiring data after a specified number of waveforms or samples are acquired. You can clear the database by turning off all three features that use the database.

The database does not differentiate waveforms from different channels or functions. If three channels are turned on and the waveform from each channel happens to light the same pixel at the same time, the counter is incremented by three. However, it is not possible to tell how many hits came from each waveform. To separate waveforms, you can set the display to two graphs or position the waveforms vertically with the channel offset. By separating the waveforms, you can avoid overlapping data in the database caused by multiple waveforms. Although multiple waveforms may be displayed in Oscilloscope mode, histogram measurements can be made on only one at a time. Set the histogram window source to the source you want to measure. Even if the display is set to show only the most recent acquisition, the database keeps track of all pixel hits while the database is building.

Remember that color-graded display, mask testing, and histograms all use the same database. Suppose that the database is building because color-graded display is ON; when mask testing or histograms are turned on, they can use the information already established in the database as though they had been turned on the entire time.

To avoid erroneous data, clear the display after you change instrument setup conditions or device under test (DUT) conditions and acquire new data before extracting measurement results.

Histogram Commands

AXIS

Command :HISTogram:AXIS {VERTical | HORizontal}

This command selects the axis of the histogram. A horizontal or vertical histogram may be created.

Example The following example defines a vertical histogram.

```
10 OUTPUT 707;”:HISTOGRAM:AXIS VERTICAL”
20 END
```

Query :HISTogram:AXIS?

The query returns the currently selected histogram axis.

Returned Format [:HISTogram:AXIS] {VERTical | HORizontal} <NL>

Example

```
10 DIM Axis$[50]
20 OUTPUT 707;”:HISTOGRAM:AXIS?”
30 ENTER 707;Axis$
40 PRINT Axis$
50 END
```

MODE

Command :HISTogram:MODE {ON | 1 | OFF | 0 | WAVeform}

This command selects the histogram mode. The histogram may be off or set to track the waveform database.

Example The following example sets the histogram mode to track the waveform database.

```
10 OUTPUT 707;”:HISTOGRAM:MODE WAVEFORM”
20 END
```

Query :HISTogram:MODE?

The query returns the currently selected histogram mode.

Returned Format [:HISTogram:MODE] {1 | 0 | WAVeform} <NL>

Example

The following example returns the result of the mode query and prints it to the controller’s screen.

```
10 DIM Mode$[10]
20 OUTPUT 707;”:HISTOGRAM:MODE?”
30 ENTER 707;Mode$
40 PRINT Mode$
50 END
```

SCALe:SIZE

Command

:HISTogram:SCALe:SIZE <size> [,{HORizontal | VERTical}]

This command sets the histogram size for vertical and horizontal mode.

<size>

The size is from 1.0 to 8.0 for the horizontal mode and from 1.0 to 10.0 for the vertical mode. Separate values are maintained for each axis. If the optional axis parameter is not specified, the size of the current axis is set.

Example

The following example sets the histogram size to 3.5.

```
10 OUTPUT 707;”:HISTOGRAM:SCALE:SIZE 3.5”
20 END
```

Query

:HISTogram:SCALe:SIZE? [HORizontal | VERTical]

The query returns the correct size of the histogram.

Returned Format

[:HISTogram:SCALe:SIZE] <size><NL>

Example

The following example returns the result of the size query and prints it to the controller’s screen.

```
10 DIM Scal$[50]
20 OUTPUT 707;”:HISTOGRAM:SCALE:SIZE?”
30 ENTER 707;Size$
40 PRINT Size$
50 END
```

WINDow:DEFault

Command

:HISTogram:WINDow:DEFault

This command positions the histogram markers to a default location on the display. Each marker will be positioned one division off the left, right, top, and bottom of the display.

Example

The following example sets the histogram window to the default position.

```
10 OUTPUT 707;”:HISTogram:WINDow:DEFault”
20 END
```

WINDow:SOURce

Command :HISTogram:WINDow:SOURce {CHANnel<N> | FUNCtion<N> | RESPonse<N> | CGRade}

This command selects the source of the histogram window. The histogram window will track the source’s vertical and horizontal scale. When color grade or gray scale data is loaded from a file, the window source is set to CGRade (color grade). No other source may be selected until the histogram database is cleared.

<N> An integer 1–4, representing the selected function.

Example The following example sets the histogram window’s source to Channel 1.

```
10 OUTPUT 707;”:HISTOGRAM:WINDOW:SOURCE CHANNEL1”
20 END
```

Query :HISTogram:WINDow:SOURce?

The query returns the currently selected histogram window source.

Returned Format [:HISTogram:WINDow:SOURce] {CHANnel<N> | FUNCtion<N> | RESPonse<N> | CGRade}<NL>

Example The following example returns the result of the window source query and prints it to the controller’s screen.

```
10 DIM Winsour$[50]
20 OUTPUT 707;”:HISTOGRAM:WINDOW:SOURCE?”
30 ENTER 707;Winsour$
40 PRINT Winsour$
50 END
```

WINDow:X1Position

Command :HISTogram:WINDow:X1Position <X1 position>

This command moves the X1 marker of the histogram window. The histogram window selects a portion of the database to histogram. The histogram window markers will track the scale of the histogram window source.

Example The following example sets the X1 position to –200 microseconds.

```
10 OUTPUT 707;”:HISTOGRAM:WINDOW:X1POSITION -200E-6”
20 END
```

Query :HISTogram:WINDow:X1Position?

The query returns the value of the X1 histogram window marker.

Returned Format [:HISTogram:WINDow:X1Position]<X1 position><NL>

Example The following example returns the result of the X1 position query and prints it to the controller’s screen.

```
10 DIM X1$[50]
20 OUTPUT 707;”:HISTOGRAM:WINDOW:X1POSITION?”
30 ENTER 707;X1$
40 PRINT X1$
50 END
```

WINDow:X2Position

Command :HISTogram:WINDow:X2Position <X2 position>

This command moves the X2 marker of the histogram window. The histogram window selects a portion of the database to histogram. The histogram window markers will track the scale of the histogram window source.

Example The following example sets the X2 marker to 200 microseconds.

```
10 OUTPUT 707;”:HISTOGRAM:WINDOW:X2POSITION 200E-6”
20 END
```

Query :HISTogram:WINDow:X2Position?

The query returns the value of the X2 histogram window marker.

Returned Format [:HISTogram:WINDow:X2Position] <X2 position><NL>

Example The following example returns the result of the X2 position query and prints it to the controller’s screen.

```
10 DIM X2$[50]
20 OUTPUT 707;”:HISTOGRAM:WINDOW:X2POSITION?”
30 ENTER 707;X2$
40 PRINT X2$
50 END
```

WINDow:Y1Position

Command

:HISTogram:WINDow:Y1Position <Y1 position>

This command moves the Y1 marker of the histogram window. The histogram window selects a portion of the database to histogram. The histogram window markers will track the scale of the histogram window source.

Example

The following example sets the position of the Y1 marker to –250 mV.

```
10 OUTPUT 707;":HISTOGRAM:WINDOW:Y1POSITION -250E-3"
20 END
```

Query

:HISTogram:WINDow:Y1Position?

The query returns the value of the Y1 histogram window marker.

Returned Format

[:HISTogram:WINDow:Y1Position] <Y1 position><NL>

Example

The following example returns the result of the Y1 position query and prints it to the controller's screen.

```
10 DIM Y1$[50]
20 OUTPUT 707;":HISTOGRAM:WINDOW:Y1POSITION?"
30 ENTER 707;Y1$
40 PRINT Y1$
50 END
```

WINDow:Y2Position

Command :HISTogram:WINDow:Y2Position <Y2 position>

This command moves the Y2 marker of the histogram window. The histogram window selects a portion of the database to histogram. The histogram window markers will track the scale of the histogram window source.

Example The following example sets the position of the Y2 marker to 1.

```
10 OUTPUT 707;":HISTOGRAM:WINDOW:Y2POSITION 1"
20 END
```

Query :HISTogram:WINDow:Y2Position?

The query returns the value of the Y2 histogram window marker.

Returned Format [:HISTogram:WINDow:Y2Position] <Y2 position><NL>

Example The following example returns the result of the Y2 position query and prints it to the controller's screen.

```
10 DIM Y2$[50]
20 OUTPUT 707;":HISTOGRAM:WINDOW:Y2POSITION?"
30 ENTER 707;Y2$
40 PRINT Y2$
50 END
```

Limit Test Commands

Limit Test Commands

The Limit Test commands and queries control the limit test features of the analyzer. Limit testing automatically compares measurement results with pass or fail limits. The limit test tracks up to four measurements. The action taken when the test fails is also controlled with commands in this subsystem.

FAIL

Command

:LTESt:FAIL {INSide | OUTSide | ALWays | NEVer}

This command sets the fail condition for an individual measurement. The conditions for a test failure are set on the source selected with the last LTESt:SOURce command. When a measurement failure is detected by the limit test, the fail action conditions are executed, and there is the potential to generate an SRQ.

INSide

FAIL:INside causes the instrument to fail a test when the measurement results are within the parameters set by the LTESt:LLIMit and LTESt:ULIMit commands.

OUTSide

FAIL:OUTside causes the instrument to fail a test when the measurement results exceed the parameters set by LTESt:LLIMit and LTESt:ULIMit commands.

ALWays

FAIL:ALWays causes the instrument to fail a test every time the measurement is executed, and the parameters set by the LTESt:LLIMit and LTESt:ULIMit commands are ignored. The FAIL:ALWays mode logs the action each time the measurement is executed. FAIL:ALWays can monitor trends in measurements, for example, tracking a measurement during an environmental test while the instrument is running a measurement for a long time, as the temperature or humidity is changed. Each time the measurement is executed, the results are logged as determined by the fail action set with the LTESt:SSCreen, LTESt:SSUMmary, or LTESt:SWAVeform commands.

NEVer

FAIL:NEVer sets the instrument so a measurement never fails a test. Use the FAIL:NEVer mode to observe one measurement but determine a failure from a different measurement. The FAIL:NEVer mode monitors a measurement without any fail criteria.

Example

The following example causes the instrument to fail a test when the measurements are outside the lower and upper limits.

```
10 OUTPUT 707;":LTEST:FAIL OUTSIDE"
20 END
```

Query

:LTESt:FAIL?

The query returns the current value set for the fail condition.

Returned Format

[:LTESt:FAIL] {INSide | OUTSide | ALWays | NEVer}<NL>

Example The following example returns the current fail condition and prints the result to the controller’s screen.

```
10 DIM FAIL$[50]
20 OUTPUT 707;”:LTEST:FAIL?”
30 ENTER 707;FAIL$
40 PRINT FAIL$
50 END
```

LLIMit

Command :LTESt:LLIMit <lower_value>

This command sets the lower test limit for the active measurement currently selected by the :LTESt:SOURce command.

<lower_value> A real number.

Example The following example sets the lower test limit to 1.

```
10 OUTPUT 707;”:LTEST:LLIMIT 1”
20 END
```

If, for example, you chose to measure volts peak-peak and want the smallest acceptable signal swing to be one volt, you could use the above command, then set the limit test to fail when the signal is outside the specified limit.

Query :LTESt:LLIMit?

The query returns the current value set by the command.

Returned Format [:LTESt:LLIMit]<lower_value><NL>

Example The following example returns the current lower test limit and prints the result to the controller’s screen.

```
10 DIM LLIM$[50]
20 OUTPUT 707;”:LTEST:LLIMIT?”
30 ENTER 707;LLIM$
40 PRINT LLIM$
50 END
```

MNFound

Command

```
:LTESt:MNFound {FAIL | PASS | IGNore}
```

This command sets the action to take when the measurement cannot be made. This command affects the active measurement currently selected by the last LTESt:SOURce command.

This command tells the instrument how to treat a measurement that cannot be made. For example, if a risetime between 1 to 5 volts is requested and the captured signal is between 2 to 3 volts, this control comes into play. Another use for this command is when trying to measure the frequency of a baseline waveform.

FAIL

FAIL is used when the instrument cannot make a measurement, for example, when an edge is expected to be present and is not found. This is the mode to use for most applications.

The total number of waveforms is incremented, and the total number of failures is incremented.

PASS

PASS might be used when triggering on one event and measuring another event which may not occur for every trigger. For example, in a communications test system, you might want to trigger on the clock and test the risetime of edges in the data stream. However, there may be no way to guarantee that a rising edge will be present to measure in the data stream at every clock edge. By using the PASS parameter, the limit test will not log a failure if there is no edge found in the data stream.

If the measurement cannot be made, the total number of waveforms measured is incremented, but the total number of failures is not.

IGNore

IGNore is similar to PASS, except the totals for the number of waveforms and failures are not incremented. Therefore, the total indicates the number of tests when the measurement was made.

Example

The following example causes the instrument to pass the test when a measurement cannot be made.

```
10 OUTPUT 707;":LTEST:MNFOUND PASS"
20 END
```

Query

```
:LTESt:MNFound?
```

The query returns the current action set by the command.

Returned Format

```
[:LTESt:MNFound] {FAIL | PASS | IGNore}<NL>
```

Example

The following example gets the current setting of the measurement not found action and prints the result to the controller's screen.

```
10 DIM MNF$[50]
20 OUTPUT 707;":LTEST:MNFOUND?"
30 ENTER 707;MNF$
40 PRINT MNF$
50 END
```

RUNTil

Command

:LTESt:RUNTil FAILures, <total_failures>

This command determines the termination conditions for the test.

Note

The keywords RUN or RUMode (Run Until Mode) may also be used. This command is compatible with the Agilent 83480/54750.

FAILures

FAILures runs the limit test until a set number of failures occur. When FAILures is sent, the test executes until the selected total failures are obtained. The number of failures are compared against this number to test for termination.

Use the FAILures mode when you want the limit test to reach completion after a set number of failures. The total number of failures is additive for all of the measurements. For example, if you select 10 failures, the total of 10 failures can come from several measurements. The 10 failures can be the sum of four rise time failures, four +width failures, and two overshoot failures.

<total_failures>

An integer: 1 to 1,000,000,000.

Example

The following example causes limit test to run until two failures occur.

```
10 OUTPUT 707;":LTEST:RUNTil FAILures, 2"
20 END
```

Query

:LTESt:RUNTil?

The query returns the currently selected termination condition and value.

Returned Format

[:LTESt:RUNTil] {FAILures, <total_failures>}<NL>

Example The following example returns the current condition under which the limit test terminates and prints the result to the controller's screen.

```
10 DIM RUN$[50]
20 OUTPUT 707;":LTEST:RUNTIL?"
30 ENTER 707;RUN$
40 PRINT RUN$
50 END
```

SOURce

Command :LTESt:SOURce {1 | 2 | 3 | 4}

This command selects the current source for the ULIMit, LLIMit, MNFound, and FAIL commands. It selects one of the active measurements as referred to by their position in the measurement window on the bottom of the screen. Source 1 is the measurement on the top line, 2 is on the second line, and so on.

Example The following example selects the first measurement as the source for the limit testing commands.

```
10 OUTPUT 707;":LTEST:SOURCE 1"
20 END
```

Query :LTESt:SOURce?

The query returns the currently selected measurement source.

Returned Format [:LTESt:SOURce] {1 | 2 | 3 | 4} <NL>

Example The following example returns the currently selected measurement source for the limit testing commands.

```
10 DIM SOURCE$[50]
20 OUTPUT 707;":LTEST:SOURCE?"
30 ENTER 707;SOURCE$
40 PRINT SOURCE$
50 END
```

See Also Measurements are started in the Measurement subsystem.

SSCReen

Command :LTESt:SSCReen {OFF | DISK [,<filename>]}

This command saves a copy of the screen in the event of a failure.

OFF Turns off the save action.

DISK Saves a copy of the screen to disk in the event of a failure.

<filename> An ASCII string enclosed in quotations marks. If no filename is specified, a filename will be assigned. The default filename is *MeasLimitScreenX.bmp*, where X is an incremental number assigned by the instrument.

If a filename is specified without a path, the default path will be C:\User Files\screen images. The default file type is a bitmap (.bmp). The following graphics formats are available by specifying a file extension: PCX files (.pcx), EPS files (.eps), Postscript files (.ps) and GIF files (.gif).

Example The following example saves a copy of the screen to the disk in the event of a failure. Additional disk-related controls are set using the SSCReen:AREA and SSCReen:IMAGe commands.

```
10 OUTPUT 707;":LTEST:SSCREEN DISK"
20 END
```

Query :LTESt:SSCReen?

The query returns the current state of the SSCReen command.

Returned Format [:LTESt:SSCReen] {OFF | DISK [,<filename>]}<NL>

Example The following example returns the destination of the save screen when a failure occurs and prints the result to the controller's screen.

```
10 DIM SSCR$[50]
20 OUTPUT 707;":LTESt:SSCREEN?"
30 ENTER 707;SSCR$
40 PRINT SSCR$
50 END
```

SSCReen:AREA

Command :LTESt:SSCReen:AREA {GRATicule | SCReen}

This command selects which data from the screen is to be saved to disk when the run until condition is met. When you select GRATicule, only the graticule area of the screen is saved (this is the same as choosing Waveforms Only in the Specify Report Action for measurement limit test dialog box). When you select SCReen, the entire screen is saved.

Example This example selects the graticule for printing.

```
10 OUTPUT 707;":LTESt:SSCReen:AREA GRATICULE"
20 END
```

Query :LTESt:SSCReen:AREA?

The query returns the current setting for the area of the screen to be saved.

Returned Format [:LTESt:SSCReen:AREA] {GRATicule | SCReen}<NL>

Example This example places the current selection for the area to be saved in the string variable, Selection$, then prints the contents of the variable to the computer's screen.

```
10 DIM Selection$[50]                    !Dimension variable
20 OUTPUT 707;":LTEST:SSCREEN:AREA?"
30 ENTER 707;Selection$
40 PRINT Selection$
50 END
```

SSCReen:IMAGe

Command `:LTESt:SSCReen:IMAGe {NORMal | INVert | MONochrome}`

This command saves the image normally, inverted, or in monochrome. IMAGe INVert is the same as choosing Invert Waveform Background in the Specify Report Action for measurement limit test dialog box.

Example This example sets the image output to normal.

```
10 OUTPUT 707;":LTESt:SSCReen:IMAGE NORMAL"
20 END
```

Query `:LTESt:SSCReen:IMAGe?`

The query returns the current image setting.

Returned Format `[:LTESt:SSCReen:IMAGe] {NORMal | INVert | MONochrome}<NL>`

Example This example places the current setting for the image in the string variable, Setting$, then prints the contents of the variable to the computer's screen.

```
10 DIM Setting$[50]                          !Dimension variable
20 OUTPUT 707;":LTEST:SSCREEN:IMAGE?"
30 ENTER 707;Setting$
40 PRINT Setting$
50 END
```

SSUMmary

Command `:LTESt:SSUMmary {OFF | DISK [,<filename>]}`

This command saves the summary in the event of a failure.

When set to disk, the summary is written to the disk drive. The summary is a logging method where the user can get an overall view of the test results. The summary is an ASCII file that the user can read on the computer or place into a spreadsheet.

<filename> An ASCII string enclosed in quotation marks. If no filename is specified, the default filename will be *MeasLimitSummaryX.sum*, where X is an incremental number assigned by the instrument. If a filename is specified without a path, the default path will be C:\User files\limit summaries.

Example The following example saves the summary to a disk file named *TEST0000.sum*.

```
10 OUTPUT 707;":LTEST:SUMMARY DISK,TEST"
20 END
```

Query :LTESt:SSUMmary?

The query returns the current specified destination for the summary.

Returned Format [:LTESt:SSUMmary] {OFF | DISK {,<filename>}}<NL>

Example The following example returns the current destination for the summary and prints the results to the controller's screen.

```
10 DIM SUMM$[50]
20 OUTPUT 707;":LTEST:SUMMARY?"
30 ENTER 707;SUMM$
40 PRINT SUMM$
50 END
```

SWAVeform

Command :LTESt:SWAVeform <source>, <destination>,[<filename>[, <format>]]

This command saves waveforms from a channel, function, TDR response or waveform memory in the event of a failure detected by the limit test. Each waveform source can be individually specified, allowing multiple channels, responses or functions to be saved to disk or waveform memories. Setting a particular source to OFF removes any waveform save action from that source.

<source> {CHANnelN | FUNCtionN | WMEMoryN | RESPonseN}

<destination> {OFF | WMEMoryN | DISK}

<filename> An ASCII string enclosed in quotation marks. If no filename is specified, the assigned filename will be *MeasLimitChN_X*, *MeasLimitFnN_X*, *MeasLimitRspN_X*, or *MeasLimitMemN_X*, where X is an incremental number assigned by the instrument. If no path is specified, the default path will be C:\User Files\waveforms.

<format> {TEXT [,YVALues | VERBose] | INTernal}

where INTernal is the default value, and VERBose is the default value for TEXT.

Example The following example turns off the saving of waveforms from channel 1 in the event of a limit test failure.

```
10 OUTPUT 707;":LTEST:SWAVEFORM CHAN1,OFF"
20 END
```

Query :LTESt:SWAVeform? <source>

The query returns the current state of the :LTESt:SWAVeform command.

Returned Format [:LTESt:SWAVeform]<source>, <destination>, [<filename>[,<format>]]<NL>

Example The following example returns the current parameters for saving waveforms in the event of a limit test failure.

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

SWAVeform:RESet

Command :LESt:SWAVeform:RESet

This command sets the save destination for all waveforms to OFF. Setting a source to OFF removes any waveform save action from that source. This is a convenient way to turn off all saved waveforms if it is unknown which are being saved.

Example

```
10 OUTPUT 707;":LEST:SWAVeform:RESet"
20 END
```

TEST

Command :LTESt:TEST {ON | 1 | OFF | 0}

This command controls the execution of the limit test function. ON allows the limit test to run over all of the active measurements. When the limit test is turned on, the limit test results are displayed on screen in a window below the graticule.

Example The following example turns off the limit test function.

```
10 OUTPUT 707;":LTEST:TEST OFF"
20 END
```

Query :LTESt:TEST?

The query returns the state of the TEST control.

Returned Format [:LTESt:TEST] {1 | 0} <NL>

Example

The following example returns the current state of the limit test (on or off, 1 or 0, respectively) and prints the result to the controller’s screen.

```
10 DIM TEST$[50]
20 OUTPUT 707;”:LTEST:TEST?”
30 ENTER 707;TEST$
40 PRINT TEST$
50 END
```

Note

The results of the MEAS:RESults? query have three extra fields when LimitTESt:TEST is ON (failures, total, status). Failures is a number, total is a number, and status is one of the following values:

0	OK
1	failed high
2	failed low
3	failed inside
4	other failures

ULIMit

Command

```
:LTESt:ULIMit <upper_value>
```

This command sets the upper test limit for the active measurement currently selected by the last :LTESt:SOURce command.

<upper_value> A real number.

Example

The following example sets the upper limit of the currently selected measurement to 500 mV.

```
10 OUTPUT 707;":LTEST:ULIMIT 500E-3"
20 END
```

Suppose you are measuring the maximum voltage of a signal with Vmax, and that voltage should not exceed 500 mV. You can use the above program and set the LTESt:FAIL OUTSide command to specify that the limit subsystem will fail a measurement when the voltage exceeds 500 mV.

Query

```
:LTESt:ULIMit?
```

The query returns the current upper limit of the limit test.

Returned Format

```
[:LTESt:ULIMit] <upper_value><NL>
```

Example

The following example returns the current upper limit of the limit test and prints the result to the controller’s screen.

```
10 DIM ULIM$[50]
20 OUTPUT 707;":LTEST:ULIMIT?"
30 ENTER 707;ULIM$
40 PRINT ULIM$
50 END
```

Marker Commands

Marker Commands

The commands in the MARKer subsystem are used to specify and query the settings of the time markers (X axis) and current measurement unit markers (volts, amps, and watts for the Y axis). The Y-axis measurement units are typically set using the CHANnel:UNITs command.

PROPagation

Command :MARKer:PROPagation {DIELectric | METer},<propagation>

This command sets the propagation velocity for TDR and TDT measurements. The propagation may be specified as a dielectric constant or in meters per second. The value is used to determine the distance from the reference plane in TDR and TDT marker measurements.

> **Note**
>
> To ensure accurate marker measurements, you must ensure that the propagation value is accurate, that the units are set correctly (:MARKer:XUNITs), and that the correct reference plane is selected (:MARKer:REFerence).

<propagation> Dielectric constant or propagation value. You must specify one of the modifiers DIELectric or METer.

Example The following example sets the propagation to 30 million meters per second.

```
10 OUTPUT 707;":MARKER:PROPAGATION METER, 3E7"
20 END
```

Query :MARKer:PROPagation?

The query returns the current propagation value.

Returned Format [:MARKer:PROPagation]<propagation> {DIELectric | METer}<NL>

Example

The following example gets the propagation value from the instrument, puts it into the variable, Prop$, then displays the contents of the variable on the controller's screen.

```
10 DIM Prop$[20]                              !Declare variable
20 OUTPUT 707;":MARKER:PROPAGATION?"
30 ENTER 707;Prop$
40 PRINT Prop$
50 END
```

REFerence

Command

:MARKer:REFerence {TRIGger | REFPlane}

This command specifies the marker reference for TDR and TDT style markers. If the reference is TRIGger, then all horizontal axis marker measurements are made with respect to the trigger point. If the reference is REFPlane, then all horizontal axis marker measurements are made with respect to the reference plane. You must perform a normalization and reference plane calibration before using a reference plane reference. This feature is available only for TDR and TDT applications.

Example

The following example sets the markers to indicate all horizontal axis measurements with respect to the trigger.

```
10 OUTPUT 707;":MARKER:REFERENCE TRIGGER"
20 END
```

Query

:MARKer:REFerence?

The query returns the current reference setting.

Returned Format

[:MARKer:REFerence] {TRIGger | REFPlane}<NL>

Example

The following example puts the current reference setting into the variable, Ref$, then displays the contents of the variable on the controller's screen.

```
10 DIM Ref$[20]                               !Declare variable
20 OUTPUT 707;":MARKER:REFERENCE?"
30 ENTER 707;Ref$
40 PRINT Ref$
50 END
```

STATe

Command :MARKer:STATe <marker_pair>,<X_marker_state>,<Y_marker_state>

This command sets the state of a marker pair.

<marker_pair> {X1Y1 | X2Y2}

Specifies which marker pair state is set.

<X_marker_state> {OFF | MANual}

Turns the X marker on or off.

<Y_marker_state> {OFF | MANual | TRACk>

Turns the Y marker off, or sets to manual placement, or sets to tracking the source waveform at the X position. TRACk is allowed only with the X_marker_state of manual. TRACk is not allowed in Eye/Mask mode.

Example This example sets the X1 marker to manual and the Y1 marker to track the source waveform at the current X1 position.

```
10 OUTPUT 707;":MARKer:STATe X1Y1, MANual, TRACk"
20 END
```

Query :MARKer:STATe? {X1Y1 | X2Y2}

Returns the states of the specified marker pair.

Returned Format [:MARKer:STATe] {X1Y1 | X2Y2},<X_marker_state>,<Y_marker_state>

Example This example returns the current state of the X2 and Y2 markers to the string variable Marker_state$, then prints the contents of the variable to the computer screen.

```
10 DIM Marker_state$[50]
20 Output 707;":MARKer:STATe? X2Y2"
30 ENTER 707;Marker_state$
40 PRINT Marker_state$
50 END
```

X1Position

Command :MARKer:X1Position <X1_position>

This command sets the X1 marker position, and moves the X1 marker to the specified time with respect to the trigger time, if the X1 marker is on.

<X1_position> Time at X1 marker in seconds.

Example This example sets the X1 marker to 90 ns.

```
10 OUTPUT 707;":MARKER:X1POSITION 90E-9"
20 END
```

Query :MARKer:X1Position?

The query returns the time at the X1 marker position.

Returned Format [:MARKer:X1Position] <X1_position><NL>

Example This example returns the current setting of the X1 marker to the numeric variable, Value, then prints the contents of the variable to the computer screen.

```
10 OUTPUT 707;":SYSTEM:HEADER OFF"      !Response headers off
20 OUTPUT 707;":MARKER:X1POSITION?"
30 ENTER 707;Value
40 PRINT Value
50 END
```

X1Y1source

Command :MARKer:X1Y1source {CHANnel<N> | FUNCtion<N> | RESPonse<N> | WMEMory<N>}

This command sets the source for the X1 and Y1 markers.

<N> For channels, functions, TDR responses and waveform memories: 1, 2, 3, or 4.

The source you specify must be enabled for markers to be displayed. If the channel, function, TDR response or waveform memory that you specify is not on, an error message is issued and the query will return NONE.

Example This example selects channel 1 as the source for markers X1 and Y1.

```
10 OUTPUT 707;":MARKER:X1Y1SOURCE CHANNEL1"
20 END
```

Query :MARKer:X1Y1source?

The query returns the current source for markers X1 and Y1.

Returned Format [:MARKer:X1Y1source] {CHANnel<N> | FUNCtion<N> | RESPonse<N> | WMEMory<N>}<NL>

Example This example returns the current source selection for the X1 and Y1 markers to the string variable, Selection$, then prints the contents of the variable to the computer screen.

```
10 DIM Selection$[50]                   !Dimension variable
20 OUTPUT 707;":MARKER:X1Y1SOURCE?"
30 ENTER 707;Selection$
40 PRINT Selection$
50 END
```

X2Position

Command :MARKer:X2Position <X2_position>

This command sets the X2 marker position and moves the X2 marker to the specified time with respect to the trigger time, if the X2 marker is on.

<X2_position> Time at X2 marker in seconds.

Example This example sets the X2 marker to 90 ns.

```
10 OUTPUT 707;":MARKER:X2POSITION 90E-9"
20 END
```

Query :MARKer:X2Position?

The query returns the time at the X2 marker in seconds.

Returned Format [:MARKer:X2Position] <X2_position><NL>

Example This example returns the current position of the X2 marker to the numeric variable, Value, then prints the contents of the variable to the computer screen.

```
10 OUTPUT 707;":SYSTEM:HEADER OFF"      !Response headers off
20 OUTPUT 707;":MARKER:X2POSITION?"
30 ENTER 707;Value
40 PRINT Value
50 END
```

X2Y2source

Command :MARKer:X2Y2source {CHANnel<N> | FUNCtion<N> | RESPonse<N> | WMEMory<N>}

This command sets the source for the X2 and Y2 markers.

<N> For channels, functions, TDR responses and waveform memories: 1, 2, 3, or 4.

The source you specify must be enabled for markers to be displayed. If the channel, function, TDR response or waveform memory that you specify is not on, an error message is issued and the query will return NONE.

Example This example selects channel 1 as the source for markers X2 and Y2.

```
10 OUTPUT 707;":MARKER:X2Y2SOURCE CHANNEL1"
20 END
```

Query :MARKer:X2Y2source?

The query returns the current source for markers X2 and Y2.

Returned Format [:MARKer:X2Y2source] {CHANnel<N> | FUNCtion<N> | RESPonse<N> | WMEMory<N>}<NL>

Example This example returns the current source selection for the X2 and Y2 markers to the string variable, Selection$, then prints the contents of the variable to the computer screen.

```
10 DIM Selection$[50]                        !Dimension variable
20 OUTPUT 707;":MARKER:X2Y2SOURCE?"
30 ENTER 707;Selection$
40 PRINT Selection$
50 END
```

XDELta?

Query :MARKer:XDELta?

This query returns the time difference between X1 and X2 time markers if they are both on. If both markers are not on, 9.999999E+37 will be returned.

Xdelta = time at X2 – time at X1

Returned Format [:MARKer:XDELta] <time><NL>

<time> Time difference between X1 and X2 time markers in seconds.

Example This example returns the current time between the X1 and X2 time markers to the numeric variable, Time, then prints the contents of the variable to the computer screen.

```
10 OUTPUT 707;":SYSTEM:HEADER OFF"      !Response headers off
20 OUTPUT 707;":MARKER:XDELTA?"
30 ENTER 707;Time
40 PRINT Time
50 END
```

XUNits

Command :MARKer:XUNITs {SECond | METer}

This command sets the units for horizontal display in TDR and TDT applications. The units may be in seconds or meters relative to the trigger or reference plane. The marker mode must be TDRTDT to use this feature. See the :MARKer:REFerence command for information on setting the reference point.

Example The following example sets the horizontal display units to meters:

```
10 OUTPUT 707;":MARKER:XUNITS METER"
20 END
```

Query :MARKer:XUNITs?

The query returns the current marker units setting.

Returned Format [:MARKer:XUNITs]{SECond | METer}<NL>

Example The following example puts the current marker units setting into the variable Units$, then displays the contents of that variable on the controller's screen.

```
10 DIM Units$[20]
20 OUTPUT 707;":MARKER:XUNITS?"
30 ENTER 707;Units$
40 PRINT Units$
50 END
```

Y1Position

Command :MARKer:Y1Position <Y1_position>

This command sets the Y1 manual marker position and moves the Y1 manual marker to the specified value on the specified source if the Y1 marker is in manual state.

<Y1_position> Current measurement unit value at Y1.

Example This example sets the Y1 marker to 10 mV.

```
10 OUTPUT 707;":MARKER:Y1POSITION 10E-3"
20 END
```

Query :MARKer:Y1Position?

The query returns the current measurement unit level at the Y1 marker position.

Returned Format [:MARKer:Y1Position] <Y1_position><NL>

Example This example returns the current setting of the Y1 marker to the numeric variable, Value, then prints the contents of the variable to the computer screen.

```
10 OUTPUT 707;":SYSTEM:HEADER OFF"      !Response headers off
20 OUTPUT 707;":MARKER:Y1POSITION?"
30 ENTER 707;Value
40 PRINT Value
50 END
```

Y2Position

Command :MARKer:Y2Position <Y2_position>

This command sets the Y2 manual marker position and moves the Y2 manual marker to the specified value on the specified source if the Y2 marker is in manual state.

<Y2_position> Current measurement unit value at Y2.

Example This example sets the Y2 marker to –100 mV.

```
10 OUTPUT 707;":MARKER:Y2POSITION -100E-3"
20 END
```

Query :MARKer:Y2Position?

The query returns the current measurement unit level at the Y2 marker position.

Returned Format [:MARKer:Y2Position] <Y2_position><NL>

Example This example returns the current setting of the Y2 marker to the numeric variable, Value, then prints the contents of the variable to the computer screen.

```
10 OUTPUT 707;":SYSTEM:HEADER OFF"     !Response headers off
20 OUTPUT 707;":MARKER:Y2POSITION?"
30 ENTER 707;Value
40 PRINT Value
50 END
```

YDELta?

Query :MARKer:YDELta?

This query returns the current measurement unit difference between Y1 and Y2 if they are both on and both have the same source. If not, 9.999999E+37 is returned.

Vdelta = value at Y2 – value at Y1

Returned Format [:MARKer:YDELta] <value><NL>

<value> Measurement unit difference between Y1 and Y2.

Example This example returns the voltage difference between Y1 and Y2 to the numeric variable, Volts, then prints the contents of the variable to the computer screen.

```
10 OUTPUT 707;":SYSTEM:HEADER OFF"     !Response headers off
20 OUTPUT 707;":MARKER:YDELTA?"
30 ENTER 707;Volts
40 PRINT Volts
50 END
```

YUNits

Command

:MARKer:YUNITs {VOLT | OHM | REFLect}

This command sets the units for vertical display in TDR and TDT applications. The units may be in volts, ohms, or % reflection. The marker mode must be TDRTDT to use this feature.

Example

The following example sets the vertical display units to ohms:

```
10 OUTPUT 707;":MARKER:YUNITS OHM"
20 END
```

Query

:MARKer:YUNITs?

This query returns the current marker units setting.

Returned Format

[:MARKer:YUNITs]{VOLT | OHM | REFLect}<NL>

Example

The following example puts the current marker units setting into the variable Units$, then displays the contents of that variable on the controller’s screen.

```
10 DIM Units$[20]
20 OUTPUT 707;":MARKER:YUNITS?"
30 ENTER 707;Units$
40 PRINT Units$
50 END
```

21

Mask Test Commands

Mask Test Commands

The Mask Test commands and queries control the mask test features. Mask testing automatically compares measurement results with the boundaries of the mask you select. Any waveform or sample that falls within the boundaries of the mask is recorded as a failure.

Note

In commands with a REGion parameter, POLYgon may be used in place of REGion for compatibility with the Agilent 83480/54750.

Mask Handling

The instrument has three features that use a specific database. This database uses a different memory area than the waveform record for each channel. The three features that use the database are histograms, mask testing, color-graded display, and gray scale. When any one of these three features is turned on, the instrument starts building the database. The database is the size of the graticule area, which is 321 pixels high by 451 pixels wide. Behind each pixel is a 16-bit counter. Each counter is incremented each time a pixel is hit by data from a channel or function. The maximum count (saturation) for each counter is 63,488. You can check to see if any of the counters is close to saturation by using the :MEASure:CGRade:PEAK? query. The color-graded display uses colors to represent the number of hits on various areas of the display.

The database continues to build until the instrument stops acquiring data or all three functions (color-graded display, mask testing, and histograms) are turned off. The instrument stops acquiring data when the power is cycled, the Stop/Single hardkey is pressed, after a specified number of waveforms or samples are acquired, or as another module is plugged in.

You can clear the database by pressing the Clear Display hardkey, cycling the power, turning off all three features that use the database, or sending a CDISplay command. The database does not differentiate waveforms from different channels or functions. If three channels are turned on and the waveform for each channel happens to light the same pixel at the same time, the counter is incremented by three. However, you cannot tell how many hits came from each waveform. For this reason, mask test is available in Eye/Mask mode only, which allows only one channel to function at a time.

To avoid erroneous data, clear the display after you change instrument setup conditions or device under test (DUT) conditions and acquire new data before extracting measurement results.

Mask Files

The analyzer provides a series of standard masks defined according to telecom and datacom standards. For a complete list of masks and templates, refer to the user's guide. You load a mask file using the DISK:LOAD or :MTESt:LOAD commands. Mask files have the *.msk* extension.

Mask Test Commands

ALIGn

Command :MTESt:ALIGn

This command automatically aligns and scales the mask to the current waveform.

Example The following example aligns the current mask to the current waveform.

```
10 OUTPUT 707;":MTEST:ALIGN"
20 END
```

AMEThod

Command :MTESt:AMEThod NRZeye

This command sets the mask alignment method to Non-Return to Zero eye (NRZeye). NRZeye is currently the only available alignment method.

This command should be used in the setup section of a mask file when defining a custom mask. It will ensure the mask will be properly aligned if more alignment methods become available in the future.

Example The following example sets the mask alignment method to NRZ.

```
10 OUTPUT 707;":MTEST:AMEThod NRZ"
20 END
```

Query :MTESt:AMEThod?

The query returns the align method, NRZ.

Returned Format [:MTESt:AMEThod] NRZ<NL>

COUNt:FAILures?

Query :MTESt:COUNt:FAILures? REGion<number>

The query returns the number of failures that occurred within a particular region. By defining regions within regions, then counting the failures for each individual region, you can implement testing at different tolerance levels for a given waveform.

The value 9.999E37 is returned if mask testing is not enabled or if you specify a region number that is not used.

<number> An integer, 1 through 8, designating the region for which you want to determine the failure count.

Returned Format [:MTESt:COUNt:FAILures] <number_of_failures><NL>

<number_of_failures> The number of failures that have occurred for the designated region.

Example The following example determines the current failure count for region 3 and prints it on the controller screen.

```
10 DIM MASK_FAILURES$[50]
20 OUTPUT 707;":MTEST:COUNT:FAILURES? REGION3"
30 ENTER 707;MASK_FAILURES$
40 PRINT MASK_FAILURES$
50 END
```

COUNt:FSAMples?

Query :MTESt:COUNt:FSAMples?

The query returns the total number of failed samples in the current mask test run. This count is for all regions and all waveforms, so if you wish to determine failures by region number, use the COUNt:FAILures? query.

The count value returned is not the sum of the failure counts for each region. For example, assume a region 2 enclosed completely by region 1. If region 1 has 100 failures, the value returned is 100, regardless of how many failures are in region 2. Because region 2 is completely enclosed, the failure count for region 2 must be less than or equal to 100 in this instance.

The value 9.999E37 is returned if mask testing is not enabled.

Returned Format [:MTESt:COUNt:FSAMples] <number_of_failed_samples><NL>

<number_of_failed_samples> The total number of failed samples for the current test run.

Example The following example determines the number of failed samples and prints the result on the controller screen.

```
10 OUTPUT 707;":SYSTEM:HEADER OFF"
20 OUTPUT 707;":MTEST:COUNT:FSAMPLES?"
30 ENTER 707;MASK_FSAMPLES
40 PRINT MASK_FSAMPLES
50 END
```

COUNt:HITS?

Query

:MTESt:COUNt:HITS? {TOTal | MARGin | MASK}

This query returns the number of failed data points (or hits) that occurred when using margin mask testing.

TOTal

Returns the total number of failed data points. For positive margins, this is the sum of the MASK and MARGin counts. For negative margins, this is the same as the MASK count.

MARGin

Returns the number of data points that occurred *between* the margin mask and the standard mask. This is the margin area. This definition is true for both positive and negative margins.

To determine a negative margin, increase the magnitude of the negative margin until the number of margin hits goes to zero. All data acquired since mask margin testing was enabled will be compared to the margin. Sampled points acquired before the margin was activated, that fall into the margin region, will also show up as mask hits.

MASK

Returns the number of data points that failed the standard mask.

Returned Format

[:MTESt:COUNt:HITS] <number_of_hits><NL>

Example

The following example determines the number of failed data points that occurred within the mask margin.

```
10 OUTPUT 707;":SYSTEM:HEADER OFF"
20 OUTPUT 707;":MTEST:COUNT:HITS? MARGin"
30 ENTER 707;Margin_hits
40 PRINT Margin_hits
50 END
```

COUNt:SAMPles?

Query

:MTESt:COUNt:SAMPles?

The query returns the total number of samples captured in the current mask test run.

The value 9.999E37 is returned if mask testing is not enabled.

Returned Format

[:MTESt:COUNt:SAMPles] <number_of_samples><NL>

<number_of _samples>

The total number of samples for the current test run.

Example

The following example determines the number of samples gathered in the current test run and prints the result on the controller screen.

```
10 OUTPUT 707;":SYSTEM:HEADER OFF"
20 OUTPUT 707;":MTEST:COUNT:SAMPLES?"
30 ENTER 707;Mask_samples
40 PRINT Mask_samples
50 END
```

COUNt:WAVeforms?

Query

:MTESt:COUNt:WAVeforms?

The query returns the total number of waveforms gathered in the current mask test run.

The value 9.999E37 is returned if mask testing is not enabled.

Returned Format

[:MTESt:COUNt:WAVeforms] <number_of_waveforms><NL>

<number_of_ waveforms>

The total number of waveforms for the current test run.

Example

The following example determines the number of waveforms gathered in the current test run and prints the result on the controller screen.

```
10 OUTPUT 707;":SYSTEM:HEADER OFF"
20 OUTPUT 707;":MTEST:COUNT:WAVEFORMS?"
30 ENTER 707;Mask_waveforms
40 PRINT Mask_waveforms
50 END
```

DELete

Command :MTESt:DELete

This command clears the currently loaded mask. MTESt:DELete is the preferred command. (See also MTESt:MASK:DELete.)

Example The following example deletes the currently defined mask.

```
10 OUTPUT 707;":MTEST:DELETE"
20 END
```

EXIT

Command :MTESt:EXIT

This command terminates mask testing.

Example The following example terminates mask testing.

```
10 OUTPUT 707;":MTEST:EXIT"
20 END
```

LOAD

> This command operates only on files and directories on "A:\", "C:\User Files", "C:\scope\masks" and any mapped network drive.

Command :MTESt:LOAD "<file_name>"

This command loads the specified mask file.

<file_name> The filename, with the extension .msk. If no file suffix is specified, .msk is appended.

You can specify the entire path, or use a relative path such as "." or ".."

If you use a relative path, the present working directory is assumed. Use DISK:CDIRectory to change the present working directory, and DISK:PWD? to query it.

If no path is specified, a search path is followed. The directory C:\User Files\masks is searched first, then C:\scope\masks.

Example This example loads the mask file *FILE1.msk*.

```
10 OUTPUT 707;":MTESt:LOAD ""FILE1.MSK"
20 END
```

MASK:DELete

Command :MTESt:MASK:DELete

This command deletes the complete currently defined mask.

Example The following example deletes the currently defined mask.

```
10 OUTPUT 707;":MTEST:MASK:DELETE”
20 END
```

> **Note**
>
> The :MTESt:MASK:DELete command performs the same function as :MTESt:DELete. The :MTESt:MASK:DELete command is provided for compatibility with the Agilent 83480/54750. For new programs, use the :MTESt:DELete form.

MMARgin:PERCent

Command :MTESt:MMARgin:PERCent <margin_percent>

This command sets the amount of mask margin to apply to the selected mask.

<margin_percent> An integer, –100 to 100, expressing the mask margin in percent.

Example The following example sets the mask margin to 50 percent.

```
10 OUTPUT 707;”:MTEST:MMARGIN:PERCENT 50”
20 END
```

Query :MTESt:MMARgin:PERCent?

The query returns the current mask margin.

Returned Format [:MTESt:MMARgin:PERCent] <margin_percent><NL>

Example The following example determines the mask margin and prints the result on the controller screen.

```
10 OUTPUT 707;":SYSTEM:HEADER OFF"
20 OUTPUT 707;":MTEST:MMARgin:PERCent?"
30 ENTER 707;Margin
40 PRINT Margin
50 END
```

MMARgin:STATe

Command :MTESt:MMARgin:STATe {ON | 1 | OFF | 0}

This command controls the activation of the mask margin.

Example The following example activates the mask margin.

```
10 OUTPUT 707;":MTEST:MMARgin:STATe ON"
20 END
```

Query :MTESt:MMARgin:STATe?

The query returns the current mask margin state.

Returned Format [:MTESt:MMARgin:STATe] {1 | 0}<NL>

Example The following example determines the mask margin state and prints the result on the controller screen.

```
10 DIM Margin_state$[50]
20 OUTPUT 707;":MTEST:MMARgin:STATe?"
30 ENTER 707;Margin_state$
40 PRINT Margin_state$
50 END
```

RUNTil

Command :MTESt:RUNTil {OFF | FSAMples, <number_of_failed_samples>}

This command selects the acquisition run until mode. The acquisition may be set to run until *n* fsamples have been acquired or to run forever (OFF). If more than one run until criteria is set, then the instrument will act upon the completion of whichever run until criteria is achieved first.

Note

The :MTESt:RUMode command serves the same function and has been retained for compatibility with the Agilent 83480/54750. All new programs should use the :RUNTil command.

<number_of_failed_ samples>	An integer from 1 to 1,000,000,000.

Example The following example specifies that the acquisition runs until 50 samples have been obtained.

```
10 OUTPUT 707;":MTESt:RUNTIL FSAMples,50"
20 END
```

Query :MTESt:RUNTil?

The query returns the currently selected run until state.

Returned Format [:MTESt:RUNTil] {OFF | FSAMPles, <n fsamples>}<NL>

Example The following example returns the result of the run until query and prints it to the controller's screen.

```
10 DIM Runt$[50]
20 OUTPUT 707;":MTESt:RUNTIL?"
30 ENTER 707;Runt$
40 PRINT Runt$
50 END
```

SCALe:DEFault

Command :MTESt:SCALe:DEFault

This command sets the scaling markers to default values. The X1, Y1, and Y2 markers are set to values corresponding to graticule positions that are two divisions in from the left, top, and bottom of the graticule, respectively.

Example The following example selects the default scale.

```
10 OUTPUT 707;":MTEST:SCALE:DEFAULT"
20 END
```

SCALe:SOURce?

Query

:MTESt:SCALe:SOURce?

The query returns the name of the source currently used to interpret the Y1 and Y2 scale factors.

Returned Format

[:MTESt:SCALe:SOURce] {WMEMory<N> | FUNCtion<N> | RESPonse<N> | CHANnel<N>} <NL>

Example

The following example gets the current scale source setting from the instrument and prints it on the controller screen.

```
10 DIM Scale_Source$[30]
20 OUTPUT 707;":MTEST:SCALE:SOURCE?"
30 ENTER 707;Scale_source$
40 PRINT Scale_source$
50 END
```

SCALe:X1

Command

:MTESt:SCALe:X1 <x1_value>

This command defines where X=0 in the base coordinate system used for mask testing. The other X coordinate is defined by the SCALe:XDELta command. Once the X1 and XDELta coordinates are set, all X values of vertices in region masks are defined with respect to this value, according to the equation:

$$X = (X \times XDELta) + X1$$

Thus, if you set X1 to 100 μs, and XDELta to 100 μs, an X value of .100 in a vertex is at 110 μs.

The instrument uses this equation to normalize vertex values. This simplifies reprogramming to handle different data rates. For example, if you halve the period of the waveform of interest, you need only to adjust the XDELta value to set up the mask for the new waveform.

<x1_value>

A time value specifying the location of the X1 coordinate, which will then be treated as X=0 for region vertex coordinates.

Example

The following example sets the X1 coordinate at 150 μs.

```
10 OUTPUT 707;":MTEST:SCALE:X1 150E-6"
20 END
```

Query

:MTESt:SCALe:X1?

The query returns the current X1 coordinate setting.

Returned Format [:MTESt:SCALe:X1] <x1_value> <NL>

Example The following example gets the current setting of the X1 coordinate from the instrument and prints it on the controller screen.

```
10 DIM Scale_x1$[50]
20 OUTPUT 707;":MTEST:SCALE:X1?"
30 ENTER 707;Scale_x1$
40 PRINT Scale_x1$
50 END
```

SCALe:XDELta

Command :MTESt:SCALe:XDELta <xdelta_value>

This command defines the position of the X2 marker with respect to the X1 marker. In the mask test coordinate system, the X1 marker defines where X=0; thus, the X2 marker defines where X=1.

Because all X vertices of regions defined for mask testing are normalized with respect to X1 and ΔX, redefining ΔX also moves those vertices to stay in the same locations with respect to X1 and ΔX. Thus, in many applications, it is best if you define XDELta as a pulse width or bit period. Then a change in data rate, without corresponding changes in the waveform, can easily be handled by changing ΔX.

The X-coordinate of region vertices are normalized using the equation:

$$X = (X \times XDELta) + X1$$

<xdelta_value> A time value specifying the distance of the X2 marker with respect to the X1 marker.

Example Assume that the period of the waveform you wish to test is 1 μs. Then the following example will set ΔX to 1 μs, ensuring that the waveform's period is between the X1 and X2 markers.

```
10 OUTPUT 707;":MTEST:SCALE:XDELTA 1E-6"
20 END
```

Query :MTESt:SCALe:XDELta?

The query returns the current value of ΔX.

Returned Format [:MTESt:SCALe:XDELta] <xdelta_value> <NL>

Example The following example gets the value of ΔX from the instrument and prints it on the controller screen.

```
10 DIM Scale_xdelta$[50]
20 OUTPUT 707;":MTEST:SCALE:XDELTA?"
30 ENTER 707;Scale_xdelta$
40 PRINT Scale_xdelta$
50 END
```

SCALe:Y1

Command :MTESt:SCALe:Y1 <y1_value>

This command defines where Y=0 in the coordinate system for mask testing. All Y values of vertices in the coordinate system are defined with respect to the boundaries set by SCALe:Y1 and SCALe:Y2, according to the equation:

$$Y = (Y \times (Y2 - Y1)) + Y1$$

Thus, if you set Y1 to 100 mV, and Y2 to 1 V, a Y value of .100 in a vertex is at 190 mV.

<y1_value> A voltage value specifying the point at which Y=0.

Example The following example sets the Y1 marker to –150 mV.

```
10 OUTPUT 707;":MTEST:SCALE:Y1 -150E-3"
20 END
```

Query :MTESt:SCALe:Y1?

The query returns the current setting of the Y1 marker.

Returned Format [:MTESt:SCALe:Y1] <y1_value><NL>

Example The following example gets the setting of the Y1 marker from the instrument and prints it on the controller screen.

```
10 DIM Scale_y1$[50]
20 OUTPUT 707;":MTEST:SCALE:Y1?"
30 ENTER 707;Scale_y1$
40 PRINT Scale_y1$
50 END
```

SCALe:Y2

Command :MTESt:SCALe:Y2 <y2_value>

This command defines the Y2 marker in the coordinate system for mask testing. All Y values of vertices in the coordinate system are defined with respect to the boundaries defined by SCALe:Y1 and SCALe:Y2, according to the following equation:

$$Y = (Y \times (Y2 - Y1)) + Y1$$

Thus, if you set Y1 to 100 mV, and Y2 to 1 V, a Y value of .100 in a vertex is at 190 mV.

<y2_value> A voltage value specifying the location of the Y2 marker.

Example The following example sets the Y2 marker to 2.5 V.

```
10 OUTPUT 707;":MTEST:SCALE:Y2 2.5"
20 END
```

Query :MTESt:SCALe:Y2?

The query returns the current setting of the Y2 marker.

Returned Format [:MTESt:SCALe:Y2] <y2_value> <NL>

Example The following example gets the setting of the Y2 marker from the instrument and prints it on the controller screen.

```
10 DIM Scale_y2$[50]
20 OUTPUT 707;":MTEST:SCALE:Y2?"
30 ENTER 707;Scale_y2$
40 PRINT Scale_y2$
50 END
```

SCALe:YTRack

Command :MTESt:SCALe:YTRack {{ON | 1} {OFF | 0}}

This command enables or disables tracking between the Y1 and Y2 levels.

Example The following program enables tracking between Y1 and Y2.

```
10 OUTPUT 707;":MTEST:SCALE:YTRACK:ON"
20 END
```

Query :MTESt:SCALe:YTRack?

The query returns the current state of the tracking.

Returned Format [:MTESt:SCALe:YTRack] {1 | 0}<NL>

Example The following example determines the state of Y tracking and prints the results on the controller screen.

```
10 DIM Ytrack_state$[50]
20 OUTPUT 707;":MTESt:SCALe:YTRack?"
30 ENTER 707;Ytrack_state$
40 PRINT Ytrack_state$
50 END
```

SSCReen

Command :MTESt:SSCReen {OFF | DISK [,<filename>]}

This command saves a copy of the screen in the event of a failure.

OFF Turns off the save action.

DISK Saves a copy of the screen to disk in the event of a failure.

<filename> An ASCII string enclosed in quotations marks. If no filename is specified, a filename will be assigned. The default filename is *MeasLimitScreenX.bmp*, where X is an incremental number assigned by the instrument.

If a filename is specified without a path, the default path will be C:\User Files\screen images. The default file type is a bitmap (.bmp). The following graphics formats are available by specifying a file extension: PCX files (.pcx), EPS files (.eps), Postscript files (.ps) and GIF files (.gif).

Example The following example saves a copy of the screen to the disk in the event of a failure. Additional disk-related controls are set using the SSCReen:AREA and SSCReen:IMAGe commands.

```
10 OUTPUT 707;":MTESt:SSCREEN DISK"
20 END
```

Query :MTESt:SSCReen?

The query returns the current state of the SSCReen command.

Returned Format [:MTESt:SSCReen] {OFF | DISK [,<filename>]}<NL>

Example The following example returns the destination of the save screen when a failure occurs and prints the result to the controller's screen.

```
10 DIM SSCR$[50]
20 OUTPUT 707;":MTESt:SSCREEN?"
30 ENTER 707;SSCR$
40 PRINT SSCR$
50 END
```

SSCReen:AREA

Command :MTESt:SSCReen:AREA {GRATicule | SCReen}

This command selects which data from the screen is to be saved to disk when the run until condition is met. When you select GRATicule, only the graticule area of the screen is saved (this is the same as choosing Waveforms Only in the Specify Report Action for mask limit test dialog box). When you select SCReen, the entire screen is saved.

Example This example selects the graticule for saving.

```
10 OUTPUT 707;":MTEST:SSCREEN:AREA GRATICULE"
20 END
```

Query :MTESt:SSCReen:AREA?

The query returns the current setting for the area of the screen to be saved.

Returned Format [:MTESt:SSCReen:AREA] {GRATicule | SCReen}<NL>

Example This example places the current selection for the area to be saved in the string variable, Selection$, then prints the contents of the variable to the computer's screen.

```
10 DIM Selection$[50]               !Dimension variable
20 OUTPUT 707;":MTEST:SSCREEN:AREA?"
30 ENTER 707;Selection$
40 PRINT Selection$
50 END
```

SSCReen:IMAGe

Command :MTESt:SSCReen:IMAGe {NORMal | INVert | MONochrome}

This command saves the screen image to disk normally, inverted, or in monochrome. IMAGe INVert is the same as choosing Invert Waveform Background Color in the Specify Report Action for acquisition limit test dialog box.

Example This example sets the image output to normal.

```
10 OUTPUT 707;":MTEST:SSCREEN:IMAGE NORMAL"
20 END
```

Query :MTESt:SSCReen:IMAGe?

The query returns the current image setting.

Returned Format [:MTESt:SSCReen:IMAGe] {NORMal | INVert | MONochrome}<NL>

Example

This example places the current setting for the image in the string variable, Setting$, then prints the contents of the variable to the computer's screen.

```
10 DIM Setting$[50]                        !Dimension variable
20 OUTPUT 707;":MTEST:SSCREEN:IMAGE?"
30 ENTER 707;Setting$
40 PRINT Setting$
50 END
```

SSUMmary

Command

:MTESt:SSUMmary {OFF | DISK [,<filename>]}

This command saves the summary in the event of a failure.

When set to disk, the summary is written to the disk drive. The summary is a logging method where the user can get an overall view of the test results. The summary is an ASCII file that the user can read on the computer or place into a spreadsheet.

<filename>

An ASCII string enclosed in quotation marks. If no filename is specified, the default filename will be *MaskLimitSummaryX.sum*, where X is an incremental number assigned by the instrument. If a filename is specified without a path, the default path will be C:\User Files\limit summaries.

Example

The following example saves the summary to a disk file named *TEST0000.sum*.

```
10 OUTPUT 707;":LTEST:SUMMARY DISK,TEST"
20 END
```

Query

:MTESt:SSUMmary?

The query returns the current specified destination for the summary.

Returned Format

[:MTESt:SSUMmary] {OFF | DISK {,<filename>}}<NL>

Example

The following example returns the current destination for the summary and prints the results to the controller's screen.

```
10 DIM SUMM$[50]
20 OUTPUT 707;":MTEST:SUMMARY?"
30 ENTER 707;SUMM$
40 PRINT SUMM$
50 END
```

STARt

Command :MTESt:STARt

This command aligns the currently loaded mask to the current waveform, and starts testing. If no mask is currently loaded, a warning message will be displayed, but no error will be generated.

SWAVeform

Command :MTESt:SWAVeform <source>, <destination>,[<filename>[, <format>]]

This command saves waveforms from a channel, function, TDR response or waveform memory in the event of a failure detected by the limit test. Each waveform source can be individually specified, allowing multiple channels, responses, or functions to be saved to disk or waveform memories. Setting a particular source to OFF removes any waveform save action from that source.

<source> {CHANnelN | FUNCtionN | RESPonseN | WMEMoryN}

<destination> {OFF | WMEMoryN | DISK}

<filename> An ASCII string enclosed in quotation marks. If no filename is specified, the assigned filename will be *MaskLimitChN_X*, *MaskLimitFnN_X*, *MaskLimitRspN_X*, or *MaskLimitMemN_X*, where X is an incremental number assigned by the instrument. If no path is specified, the default path will be C:\User Files\waveforms.

<format> {TEXT [,YVALues | VERBose] | INTernal}

where INTernal is the default value, and VERBose is the default value for TEXT.

Example The following example turns off the saving of waveforms from channel 1 in the event of a limit test failure.

```
10 OUTPUT 707;":MTEST:SWAVEFORM CHAN1,OFF"
20 END
```

Query :MTESt:SWAVeform? <source>

The query returns the current state of the :MTESt:SWAVeform command.

Returned Format [:MTESt:SWAVeform] <source>, <destination>, [<filename>[,<format>]]<NL>

Example The following example returns the current parameters for saving waveforms in the event of a limit test failure.

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

SWAVeform:RESet

Command :MTESt:SWAVeform:RESet

This command sets the save destination for all waveforms to OFF. Setting a source to OFF removes any waveform save action from that source. This is a convenient way to turn off all saved waveforms if it is unknown which are being saved.

Example

```
10 OUTPUT 707;":MTEST:SWAVeform:RESet"
20 END
```

TEST

Command :MTESt:TEST {ON | 1 | OFF | 0}

This command controls the execution of the Mask Test function. ON allows the Mask Test to run for the active source. When the Mask Test is turned on, the Mask Test results are displayed on screen in a window below the graticule in the mask test window. OFF disables mask testing.

Example The following example determines whether the mask test subsystem is on or off and prints the result on the controller screen.

```
10 DIM Mtest_state$[30]
20 OUTPUT 707;":MTEST:TEST?"
30 ENTER 707;Mtest_state$
40 PRINT Mtest_state$
50 END
```

Query :MTESt:TEST?

The query returns the state of the mask test subsystem, whether on or off.

Returned Format [:MTESt:TEST] {1 | 0}<NL>

TITLe?

Query :MTESt:TITLe?

This query returns the string of the currently loaded mask. If no mask is loaded, a null string is returned.

Returned Format [:MTESt:TITLe] <“title”>

Measure Commands

Measure Commands

The commands in the MEASure subsystem are used to make parametric measurements on displayed waveforms.

The Agilent 86100A has three modes: Eye, TDR, and Oscilloscope. Each mode has a set of measurements. The TDR mode is not supported at this time.

In Eye mode, all of the measurements are made on the color grade/gray scale data, with the exception of average optical power and histogram measurements.

Measurement Setup

To make a measurement, the portion of the waveform required for that measurement must be displayed on the analyzer.

- For a period or frequency measurement, at least one and one half complete cycles must be displayed.
- For a pulse width measurement, the entire pulse must be displayed.
- For a rise time measurement, the leading (positive-going) edge of the waveform must be displayed.
- For a fall time measurement, the trailing (negative-going) edge of the waveform must be displayed.
- A valid source for the measurement must be designated. This can be done globally with the MEASure:SOURce command or locally with the optical source parameter in each measurement.

User-Defined Measurements

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

Measurement Error

If a measurement cannot be made because of the lack of data, because the source signal is not displayed, the requested measurement is not possible (for example, a period measurement on an FFT waveform), or for some other reason, the following results are returned:

- 9.99999E+37 is returned as the measurement result.
- If SENDvalid is ON, the error code is also returned.

Making Measurements

If more than one period, edge, or pulse is displayed, time measurements are made on the first, left-most portion of the displayed waveform.

When any of the defined measurements are requested, the analyzer first determines the top (100%) and base (0%) voltages of the waveform. From this information, the analyzer determines the other important voltage values (10%, 90%, and 50% voltage values) for making measurements.

The 10% and 90% voltage values are used in the rise-time and fall-time 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.

You can also make measurements using user-defined parameters, instead of the standard measurement values.

When the command form of a measurement is used, the analyzer is placed in the continuous measurement mode. The measurement result will be displayed on the front panel. There may be a maximum of four measurements running continuously. Use the SCRatch command to turn off the measurements.

When the query form of the measurement is used, the measurement is made one time, and the measurement result is returned.

- If the current acquisition is complete, the current acquisition is measured and the result is returned.
- If the current acquisition is incomplete and the analyzer is running, acquisitions will continue to occur until the acquisition is complete. The acquisition will then be measured and the result returned.
- If the current acquisition is incomplete and the analyzer is stopped, the measurement result will be 9.99999E+37 and the incomplete result state will be returned if SENDvalid is ON.

All measurements are made using the entire display, except for VRMS which allows measurements on a single cycle, and eye measurements in the defined eye window. Therefore, if you want to make measurements on a particular cycle, display only that cycle on the screen.

Measurements are made on the displayed waveforms specified by the SOURce command. The SOURce command allows two sources to be specified. Most measurements are only made on a single source. Some measurements, such as the DELTatime measurement, require two sources.

The measurement source for remote measurements can not be set from the front panel. The measurement source is not reset by power cycles or default setup.

If the signal is clipped, the measurement result may be questionable. In this case, the value returned is the most accurate value that can be made using the current scaling. You might be able to obtain a more accurate measurement by adjusting the vertical scale to prevent the signal from being clipped. The measurement result 9.99999E+37 may be returned in some cases of clipped signals.

Measure Commands

ANNotation

Command :MEASure:ANNotation {ON | 1 | OFF | 0}

This command turns measurement annotations on or off. If there are no active measurements, you can still turn on or off measurement annotations. The instrument will remain in the defined state and will be activated (if on) the next time measurements are performed.

Mode All instrument modes

Example The following example turns on measurement annotations.

```
10 OUTPUT 707;":MEASURE:ANNOTATION ON"
20 END
```

Query :MEASure:ANNotation?

The query returns the current measurement annotation state.

Returned Format [:MEASure:ANNotation] {1 | 0}

APOWer

Command :MEASure:APOWer <units> [,<source>]

This command measures the average power. Sources are specified with the MEASure:SOURce command or with the optional parameter following the APOWer command. The average optical power can only be measured on an optical channel input.

Mode Eye or Oscilloscope modes

<units> {WATT | DECibel}

<source> {CHANnel<number>}

<number> For channels, this value is dependent on the type of module and its location in the instrument. It will work only on optical channels.

Example The following example measures the average power of the last specified signal.

```
10 OUTPUT 707;":MEASURE:APOWER WATT"
20 END
```

Query :MEASure:APOWer? <units> [,<source>]

The query returns the measured power of the specified source.

Returned Format [:MEASure:APOWer] <value>[,<result_state>]<NL>

<value> The average power.

<result_state> If SENDvalid is ON, the result state is returned with the measurement result. Refer to the MEASure:RESults command for a list of the result states.

Example The following example places the current power of the specified signal in the numeric variable, Value, then prints the contents of the variable to the controller screen.

```
10 OUTPUT 707;":SYSTEM:HEADER OFF"          !Response headers off
20 OUTPUT 707;":MEASURE:APOWER? WATT"
30 ENTER 707;Value
40 PRINT Value
50 END
```

CGRade:AMPLitude

Command :MEASure:CGRade:AMPLitude

This command measures the eye amplitude of the displayed source. The eye amplitude is the difference between the one level and the zero level.

Mode Eye mode only

Example The following example measures the eye amplitude of the displayed signal.

```
10 OUTPUT 707;":MEASURE:CGRADE:AMPLITUDE"
20 END
```

Query :MEASure:CGRade:AMPLitude?

The query returns the eye amplitude of the eye signal of the displayed source.

Returned Format [:MEASure:CGRade:AMPLitude] <value>[<result_state>]<NL>

<value> The eye amplitude.

<result state> Refer to "RESults?" on page 22-31 for a description of result states.

Example This example queries the analyzer for the eye amplitude of the displayed signal, places the result in the numeric variable, EyeAmp, and then prints the contents of the variable to the controller's screen.

```
10 OUTPUT 707;":SYSTEM:HEADER OFF"          !Response headers off
20 OUTPUT 707;":MEASURE:CGRADE:AMPLITUDE?"
30 ENTER 707;EyeAmp
40 PRINT EyeAmp
50 END
```

CGRade:BITRate

Command :MEASure:CGRade:BITRate

This command measures the bit rate of the displayed signal. The bit rate is the number of bits per second. It is measured as the inverse of the bit period. The bit period is the time interval between two successive crossing points of an eye.

Mode Eye mode only

Example The following example measures the bit rate of the displayed eye.

```
10 OUTPUT 707;":MEASURE:CGRADE:BITRATE"
20 END
```

Query :MEASure:CGRade:BITRate?

The query returns the bit rate of the eye signal of the displayed source. Units are in bits/s.

Returned Format [:MEASure:CGRade:BITRate] <value>[<result_state>]<NL>

<value> The bit rate.

<result_state> Refer to “RESults?” on page 22-31 for a description of result states.

Example This example queries the analyzer for the bit rate of the displayed signal, places the result in the numeric variable, BitRate, and then prints the contents of the variable to the controller’s screen.

```
10 OUTPUT 707;":SYSTEM:HEADER OFF"          !Response headers off
20 OUTPUT 707;":MEASURE:CGRADE:BITRATE?"
30 ENTER 707;BitRate
40 PRINT BitRate
50 END
```

CGRade:COMPlete

Command :MEASure:CGRade:COMPlete <comp_hits>

This command sets the color grade measurement completion criterion. The data for color grade display is the same as for gray scale display.

Mode Eye or Oscilloscope modes

<comp_hits> The number of hits that the peak-numbers-of-hits, in the color grade database, must equal or exceed before a color grade measurement is complete.

Example The following example sets the completion criterion to 10 hits.

```
10 OUTPUT 707;":MEASURE:CGRADE:COMPLETE 10"
20 END
```

Query :MEASure:CGRade:COMPlete?

The query returns the current setting for color grade completion.

Returned Format [:MEASure:CGRade:COMPlete] <comp_hits><NL>

A color grade measurement query will return 9.99999E+37 until the measurement is complete.

Example The following example sets the color grade complete value, then starts a Vmax measurement with the color grade database as the source.

```
10 OUTPUT 707;":MEASURE:CGRADE:COMPLETE? 8"
20 OUTPUT 707;":DISPLAY:CGRADE ON"
30 OUTPUT 707;":MEASURE:VMAX CGRADE"
40 END
```

CGRade:CROSsing

Command :MEASure:CGRade:CROSsing

This command measures the crossing level percent of the current eye diagram on the color grade or gray scale display. The data for color grade display is the same as for gray scale display.

Mode Eye mode only

Example The following example measures the crossing level.

```
10 OUTPUT 707;":MEASURE:CGRade:CROSsing"
20 END
```

Query :MEASure:CGRade:CROSsing?

The query returns the crossing level percent of the current eye diagram on the color grade or gray scale display.

Returned Format [:MEASure:CGRade:CROSsing] <value>[,<result_state>]<NL>

<value> The crossing level.

<result_state> If SENDvalid is ON, the result state is returned with the measurement result. Refer to the MEASure:RESults command for a list of the result states.

Example The following example places the current crossing level in the numeric variable, Value, then prints the contents of the variable to the controller screen.

```
10 OUTPUT 707;":SYSTEM:HEADER OFF"                    !Response headers off
20 OUTPUT 707;":MEASURE:CGRADE:CROSSING?"
30 ENTER 707;Value
40 PRINT Value
50 END
```

CGRade:DCDistortion

Command :MEASure:CGRade:DCDistortion <format>

This command measures the duty cycle distortion on the eye diagram of the current color grade or gray scale display. The parameter specifies the format for reporting the measurement. The data for color grade display is the same as for gray scale display.

Mode Eye mode only

<format> {TIME | PERCent}

Example The following example measures the duty cycle distortion.

```
10 OUTPUT 707;":MEASURE:CGRADE:DCDistortion TIME"
20 END
```

Query :MEASure:CGRade:DCDistortion? <format>

The query returns the duty cycle distortion of the color grade or gray scale display.

Returned Format [:MEASure:CGRade:DCDistortion] <value>[,<result_state>] <NL>

<value> The duty cycle distortion.

<result_state> If SENDvalid is ON, the result state is returned with the measurement result. Refer to the MEASure:RESults command for a list of the result states.

Example The following example places the current duty cycle distortion in the numeric variable, Value, then prints the contents of the variable to the controller screen.

```
10 OUTPUT 707;":SYSTEM:HEADER OFF"                    !Response headers off
20 OUTPUT 707;":MEASURE:CGRADE:DCDISTORTION? PERCENT"
30 ENTER 707;Value
40 PRINT Value
50 END
```

CGRade:EHEight

Command :MEASure:CGRade:EHEight

This command measures the eye height on the eye diagram of the current color grade display. The data for color grade display is the same as for gray scale display.

Mode Eye mode only

Example The following example measures the eye height.

```
10 OUTPUT 707;":MEASURE:CGRADE:EHEight"
20 END
```

Query :MEASure:CGRade:EHEight?

The query returns the eye height of the color grade display.

Returned Format [:MEASure:CGRade:EHEight] <value>[,<result_state>]<NL>

<value> The eye height.

<result_state> If SENDvalid is ON, the result state is returned with the measurement result. Refer to the MEASure:RESults command for a list of the result states.

Example The following example places the current eye height in the numeric variable, Value, then prints the contents of the variable to the controller screen.

```
10 OUTPUT 707;":SYSTEM:HEADER OFF"              !Response headers off
20 OUTPUT 707;":MEASURE:CGRADE:EHEIGHT?"
30 ENTER 707;Value
40 PRINT Value
50 END
```

CGRade:ERATio

Command :MEASure:CGRade:ERATio <format>

This command measures the extinction ratio on the eye diagram of the current color grade display. The dark level or dc offset of the input channel must have been previously calibrated. The data for color grade display is the same as for gray scale display.

Mode Eye mode only

<format> {RATio | DECibel | PERCent}

Example The following example measures the extinction ratio.

```
10 OUTPUT 707;":MEASURE:CGRADE:ERATIO RATIO"
20 END
```

Query :MEASure:CGRade:ERATio? <format>

The query returns the extinction ratio of the color grade display.

Returned Format [:MEASure:CGRade:ERATio] <value>[,<result_state>]<NL>

<value> The extinction ratio.

<result_state> If SENDvalid is ON, the result state is returned with the measurement result. Refer to the MEASure:RESults command for a list of the result states.

Example The following example places the current extinction ratio in the numeric variable, Value, then prints the contents of the variable to the controller screen.

```
10 OUTPUT 707;":SYSTEM:HEADER OFF"                    !Response headers off
20 OUTPUT 707;":MEASURE:CGRADE:ERATIO? RATIO"
30 ENTER 707;Value
40 PRINT Value
50 END
```

CGRade:ESN

Command :MEASure:CGRade:ESN

This command measures the eye signal-to-noise. The data for color grade display is the same as for gray scale display.

Note: This measurement was called Q-factor in the 83480A/84750A.

Mode Eye mode only

Example The following example measures the eye signal-to-noise.

```
10 OUTPUT 707;":MEASURE:CGRADE:ESN"
20 END
```

Query :MEASure:CGRade:ESN?

The query returns the eye signal-to-noise of the color grade display.

Returned Format [:MEASure:CGRade:ESN] <value>[,<result_state>]<NL>

<value> The eye signal-to-noise value.

<result_state> If SENDvalid is ON, the result state is returned with the measurement result. Refer to the MEASure:RESults command for a list of the result states.

Example

The following example places the eye signal-to-noise value in the numeric variable, Value, then prints the contents of the variable to the controller screen.

```
10 OUTPUT 707;":SYSTEM:HEADER OFF"                    !Response headers off
20 OUTPUT 707;":MEASURE:CGRADE:ESN?"
30 ENTER 707;Value
40 PRINT Value
50 END
```

CGRade:EWIDth

Command

:MEASure:CGRade:EWIDth[<format>]

This command measures the eye width on the eye diagram of the current color grade display. The data for color grade display is the same as for gray scale display.

Mode

Eye mode only

<format>

{RATio | TIME}

The default format is TIME.

Example

The following example measures the eye width.

```
10 OUTPUT 707;":MEASURE:CGRADE:EWIDTH"
20 END
```

Query

:MEASure:CGRade:EWIDth?

The query returns the eye width of the color grade display.

Returned Format

[:MEASure:CGRade:EWIDth] <value>[,<result_state>] <NL>

<value>

The eye width.

<result_state>

If SENDvalid is ON, the result state is returned with the measurement result. Refer to the MEASure:RESults command for a list of the result states.

Example

The following example places the current eye width in the numeric variable, Value, then prints the contents of the variable to the controller screen.

```
10 OUTPUT 707;":SYSTEM:HEADER OFF"                    !Response headers off
20 OUTPUT 707;":MEASURE:CGRADE:EWIDTH?"
30 ENTER 707;Value
40 PRINT Value
50 END
```

CGRade:JITTer

Command :MEASure:CGRade:JITTer <format>

This command measures the jitter at the eye diagram crossing point. The parameter specifies the format, peak-to-peak or RMS, in which the results are reported. The data for color grade display is the same as for gray scale display.

Mode Eye or Oscilloscope modes. In either mode the source is color grade data.

<format> {PP | RMS}

Example The following example measures the jitter.

```
10 OUTPUT 707;":MEASURE:CGRADE:JITTER RMS"
20 END
```

Query :MEASure:CGRade:JITTer? <format>

The query returns the jitter of the color grade display.

Returned Format [:MEASure:CGRade:JITTer] <value>[,<result_state>] <NL>

<value> The jitter.

<result_state> If SENDvalid is ON, the result state is returned with the measurement result. Refer to the MEASure:RESults command for a list of the result states.

Example The following example places the current jitter in the numeric variable, Value, then prints the contents of the variable to the controller screen.

```
10 OUTPUT 707;":SYSTEM:HEADER OFF"                !Response headers off
20 OUTPUT 707;":MEASURE:CGRADE:JITTER? RMS"
30 ENTER 707;Value
40 PRINT Value
50 END
```

CGRade:OLEVel

Command :MEASure:CGRade:OLEVel

This command measures the logic one level inside the eye window.

Mode Eye mode only

Example The following example measures the logic one level.

```
10 OUTPUT 707;":MEASURE:CGRADE:OLEVEL"
20 END
```

Query :MEASure:CGRade:OLEVel?

The query returns the logic one level of the color grade display.

Returned Format	[:MEASure:CGRade:OLEVel] <value>[,<result_state>]<NL>
<value>	The logic one level.
<result_state>	If SENDvalid is ON, the result state is returned with the measurement result. Refer to the MEASure:RESults command for a list of the result states.
Example	The following example places the current logic one level in the numeric variable, Value, then prints the contents of the variable to the controller screen.

```
10 OUTPUT 707;":SYSTEM:HEADER OFF"                    !Response headers off
20 OUTPUT 707;":MEASURE:CGRADE:OLEVEL?"
30 ENTER 707;Value
40 PRINT Value
50 END
```

CGRade:PEAK?

Query	:MEASure:CGRade:PEAK?
	The query returns the maximum number of hits of the color grade display. The data for color grade display is the same as for gray scale display.
Mode	Eye or Oscilloscope modes
Returned Format	[:MEASure:CGRade:PEAK] <value>[,<result_state>]<NL>
<value>	The number of hits.
<result_state>	If SENDvalid is ON, the result state is returned with the measurement result. Refer to the MEASure:RESults command for a list of the result states.
Example	The following example places the current number of hits in the numeric variable, Value, then prints the contents of the variable to the controller screen.

```
10 OUTPUT 707;":SYSTEM:HEADER OFF"                    !Response headers off
20 OUTPUT 707;":MEASURE:CGRADE:PEAK?"
30 ENTER 707;Value
40 PRINT Value
50 END
```

CGRade:ZLEVel

Command	:MEASure:CGRade:ZLEvel
	This command measures logic zero level inside the eye window on the eye diagram of the current color grade display.
Mode	Eye mode only

Example The following example measures the logic zero level.

```
10 OUTPUT 707;":MEASURE:CGRADE:ZLEVel"
20 END
```

Query :MEASure:CGRade:ZLEVel?

The query returns the logic zero level of the color grade display.

Returned Format [:MEASure:CGRade:ZLEVel] <value>[,<result_state>]<NL>

<value> The logic zero level.

<result_state> If SENDvalid is ON, the result state is returned with the measurement result. Refer to the MEASure:RESults command for a list of the result states.

Example The following example places the current logic zero level in the numeric variable, Value, then prints the contents of the variable to the controller screen.

```
10 OUTPUT 707;":SYSTEM:HEADER OFF"                    !Response headers off
20 OUTPUT 707;":MEASURE:CGRADE:ZLEVel?"
30 ENTER 707;Value
40 PRINT Value
50 END
```

CLEar

Command :MEASure:CLEar

This command clears the measurement results from the screen. It is identical to the :MEASure:SCRatch command.

Example The following example clears the current measurement results from the screen.

```
10 OUTPUT 707;":MEASURE:CLEAR"
20 END
```

DEFine

Command :MEASure:DEFine <meas_spec>

This command sets up the definition for measurements by specifying the delta time, threshold, or top-base values. Changing these values may affect other measure commands. The following table identifies the relationships between user-DEFined values and other MEASure commands.

<meas_spec> {THResholds,TOPBase,EWINdow}

Table 22-1. :MEASure:DEFine Interactions

MEASure Commands	THResholds	TOPBase	EWINdow
RISEtime	x	x	
FALLtime	x	x	
PERiod	x	x	
FREQuency	x	x	
VTOP		x	
VBASe		x	
VAMPlitude		x	
PWIDth	x	x	
NWIDth	x	x	
OVERshoot	x	x	
DUTycycle	x	x	
DELTatime	x	x	
VRMS	x	x	
PREShoot	x	x	
VLOWer	x	x	
VMIDdle	x	x	
VUPPer	x	x	
VAVerage	x	x	
VARea	x	x	
CGRade:DCDistortion	x		
CGRade:CROSsing			x
CGRade:ERATio			x
CGRade:EHEight			x
CGRade:ESN			x
CGRade:OLEVel			x
CGRade:ZLEVel			x

Command

```
:MEASure:DEFine THResholds,{{STANdard} |
{PERCent,<upper_pct>,<middle_pct>,<lower_pct>} |
{UNITs,<upper_volts>,<middle_volts>,<lower_volts>}}
```

<upper_pct>
<middle_pct>
<lower_pct>

An integer, –25 to 125.

<upper_units>
<middle_units>
<lower_units>

A real number specifying amplitude units.

Command :MEASure:DEFine TOPBase,{{STANdard} |{<top_volts>,<base_volts>}}

<top_volts> <base_volts> A real number specifying voltage.

Command :MEASure:DEFine EWINdow,<ewind1pct>,<ewind2pct>

<ewind1pct> <ewind2pct> An integer, 0 to 100, specifying an eye window as a percentage of the bit period unit interval.

Example 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.

Source is specified either using MEASure:SOURce, or using the optional <source> parameter when the DELTatime measurement is started.

Query :MEASure:DEFine? {EWINdow | THResholds | TOPBase}

Returned Format [:MEASure:DEFine] CGRade,<signal_type><NL>

[:MEASure:DEFine] THResholds {{STANdard} |
{PERcent,<upper_pct>,<middle_pct>,<lower_pct>} |
{VOLTage, <upper_volts>,<middle_volts>,<lower_volts>}}<NL>

[:MEASure:DEFine] TOPBase {{STANdard} |{<top_volts>,<base_volts>}}<NL>

Use the Suffix Multiplier Instead

Using "mV" or "V" following the numeric value for the voltage value will cause Error 138-Suffix not allowed. Instead, use the convention for the suffix multiplier as described in Chapter 3, “Message Communication and System Functions”.

Example This example returns the current setup for the measurement thresholds to the string variable, Setup$, then prints the contents of the variable to the computer's screen.

```
10 DIM Setup$[50]                    !Dimension variable
20 OUTPUT 707;":MEASURE:DEFINE? THRESHOLDS"
30 ENTER 707; Setup$
40 PRINT Setup$
50 END
```

DUTYcycle

Command :MEASure:DUTYcycle [<source>]

Measures the ratio of the positive pulse width to the period. Sources are specified with the MEASure:SOURce command or with the optional parameter following the DUTYcycle command.

Mode	Oscilloscope mode only
<source>	{CHANnel<N> | FUNCtion<N> | RESPonse<N> | WMEMory<N>}
<N>	For channels: Value is dependent on the type of plug-in and its location in the instrument. For functions: 1 or 2. For waveform memories (WMEMORY): 1, 2, 3, or 4. For TDR responses: 1, 2, 3, or 4.

Example The following example measures the duty cycle of the last specified signal.

```
10 OUTPUT 707;":MEASURE:DUTYCYCLE"
20 END
```

Query :MEASure:DUTYcycle? [<source>]

The query returns the measured duty cycle of the specified source.

Returned Format [:MEASure:DUTYcycle] <value>[,<result_state>]<NL>

<value> The ratio of the positive pulse width to the period.

<result_state> If SENDVALID is ON, the result state is returned with the measurement result. Refer to the MEASure:RESults command, for a list of the result states.

Example The following example places the current duty cycle of the specified signal in the numeric variable, Value, then prints the contents of the variable to the controller's screen.

```
10 OUTPUT 707;":SYSTEM:HEADER OFF" !Response headers off
20 OUTPUT 707;":MEASURE:DUTYCYCLE?"
30 ENTER 707;Value
40 PRINT Value
50 END
```

FALLtime

Command :MEASure:FALLtime [<source>]

This command measures the time at the upper threshold of the falling edge, measures the time at the lower threshold of the falling edge, then calculates the fall time. Sources are specified with the MEASure:SOURce command or with the optional parameter following the FALLtime command.

The first displayed falling edge is used for the fall-time measurement. Therefore, for best measurement accuracy, set the sweep speed as fast as possible while leaving the falling edge of the waveform on the display.

Fall time = time at lower threshold point – time at upper threshold point.

Mode Eye and Oscilloscope modes

<source> {CHANnel<N> | FUNCtion<N> | RESPonse<N> | WMEMory<N> | CGRade}

Where CHANnel<N>, FUNCtion<N>, RESPonse<N> and WMEMory<N> apply in Oscilloscope mode only, and CGRade in Eye mode only.

<N> For channels, functions, TDR responses and waveform memories: 1, 2, 3, or 4.

Example This example measures the fall time of the last specified signal.

```
10 OUTPUT 707;":MEASURE:FALLTIME"
20 END
```

Query :MEASure:FALLtime?[<source>]

The query returns the fall time of the specified source.

Returned Format [:MEASure:FALLtime] <value>[,<result_state>]<NL>

<value> Time at lower threshold – time at upper threshold.

<result_state> If SENDvalid is ON, the result state is returned with the measurement result. Refer to the MEASure:RESults command for a list of the result states.

Example This example places the current value for fall time in the numeric variable, Value, then prints the contents of the variable to the computer's screen.

```
10 OUTPUT 707;":SYSTEM:HEADER OFF"      !Response headers off
20 OUTPUT 707;":MEASURE:FALLTIME?"
30 ENTER 707;Value
40 PRINT Value
50 END
```

FREQuency

Command :MEASure:FREQuency [<source>]

Measures the frequency of the first complete cycle on the screen using the mid-threshold levels of the waveform (50% levels if standard measurements are selected). The source is specified with the MEASure:SOURce command or with the optional parameter following the FREQuency command.

The algorithm is:

If the first edge on screen 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).

Mode Oscilloscope mode only

<source> {CHANnel<N> | FUNCtion<N> | RESPonse<N> | WMEMory<N>}

<N>	For channels: Value is dependent on the type of plug-in and its location in the instrument. For functions: 1 or 2. For waveform memories (WMEMORY): 1, 2, 3, or 4. For TDR responses: 1, 2, 3, or 4.

Example The following example measures the frequency of the last specified signal.

```
10 OUTPUT 707;":MEASURE:FREQUENCY"
20 END
```

Query :MEASure:FREQuency? [<source>]

The query returns the measured frequency.

Returned Format [:MEASure:FREQuency] <value>[,<result_state>]<NL>

<value> The frequency value, in Hertz, of the first complete cycle on the screen using the mid-threshold levels of the waveform.

<result_state> If SENDVALID is ON, the result state is returned with the measurement result. Refer to the MEASure:RESults command, for a list of the result states.

Example The following example places the current frequency of the signal in the numeric variable, Freq, then prints the contents of the variable to the controller's screen.

```
10 OUTPUT 707;":SYSTEM:HEADER OFF" !Response headers off
20 OUTPUT 707;":MEASURE:FREQUENCY?"
30 ENTER 707;Freq
40 PRINT Freq
50 END
```

HISTogram:HITS?

Query :MEASURE:HISTogram:HITS? [<source>]

This query returns the number of hits within the histogram. The source can be specified with the optional parameter following the HITS query. The HISTogram:HITS? query only applies to the histogram.

<source> {HISTogram}

Returned Format [:MEASure:HISTogram:HITS] <value>[,<result_state>]<NL>

<value> The number of hits in the histogram.

<result_state> If SENDvalid is ON, the result state is returned with the measurement result. Refer to the MEASure:RESults command for a list of the result states.

Example The following example returns the number of hits within the current histogram and prints the result to the controller screen.

```
10 OUTPUT 707;":SYSTEM:HEADER OFF"            !Response headers off
20 OUTPUT 707;":MEASURE:HISTOGRAM:HITS?"
30 ENTER 707;Histhits
40 PRINT Histhits
50 END
```

HISTogram:M1S?

Query :MEASURE:HISTogram:M1S? [<source>]

This query returns the percentage of points that are within one standard deviation of the mean of the histogram. The source can be specified with the optional parameter following the M1S query. The HISTogram:M1S? query only applies to the histogram waveform.

<source> {HISTogram}

Returned Format [:MEASure:HISTogram:M1S] <value>[,<result_state>]<NL>

<value> The percentage of points within one standard deviation of the mean of the histogram.

<result_state> If SENDvalid is ON, the result state is returned with the measurement result. Refer to the MEASure:RESults command for a list of the result states.

Example The following example returns the percentage of points within one standard deviation of the mean of the current histogram and prints the result to the controller screen.

```
10 OUTPUT 707;":SYSTEM:HEADER OFF"            !Response headers off
20 OUTPUT 707;":MEASURE:HISTOGRAM:M1S?"
30 ENTER 707;Histm1s
40 PRINT Histm1s
50 END
```

HISTogram:M2S?

Query :MEASURE:HISTogram:M2S? [<source>]

This query returns the percentage of points that are within two standard deviations of the mean of the histogram. The sources can be specified with the optional parameter following the M2S query. The HISTogram:M2S? query only applies to the histogram waveform.

<source> {HISTogram}

Returned Format [:MEASure:HISTogram:M2S] <value>[,<result_state>]<NL>

<value> The percent of points within two standard deviations of the mean of the histogram.

<result_state> If SENDvalid is ON, the result state is returned with the measurement result. Refer to the MEASure:RESults command for a list of the result states.

Example The following example returns the percentage of points within two standard deviations of the mean of the current histogram and prints the result to the controller screen.

```
10 OUTPUT 707;":SYSTEM:HEADER OFF"          !Response headers off
20 OUTPUT 707;":MEASURE:HISTOGRAM:M2S?"
30 ENTER 707;Histm2s
40 PRINT Histm2s
50 END
```

HISTogram:M3S?

Query :MEASURE:HISTogram:M3S? [<source>]

This query returns the percentage of points that are within three standard deviations of the mean of the histogram. The source can be specified with the optional parameter following the M3S query. The HISTogram:M3S? query only applies to the histogram waveform.

<source> {HISTogram}

Returned Format [:MEASure:HISTogram:M3S] <value>[,<result_state>] <NL>

<value> The percentage of points within three standard deviations of the mean of the histogram.

<result_state> If SENDvalid is ON, the result state is returned with the measurement result. Refer to the MEASure:RESults command for a list of the result states.

Example The following example returns the percentage of points within three standard deviations of the mean of the current histogram and prints the result to the controller screen.

```
10 OUTPUT 707;":SYSTEM:HEADER OFF"          !Response headers off
20 OUTPUT 707;":MEASURE:HISTOGRAM:M3S?"
30 ENTER 707;Histm3s
40 PRINT Histm3s
50 END
```

HISTogram:MEAN?

Query :MEASURE:HISTogram:MEAN? [<source>]

This query returns the mean of the histogram. The mean of the histogram is the average value of all the points in the histogram. The source can be specified with the optional parameter following the MEAN query. The HISTogram:MEAN? query only applies to the histogram waveform.

<source> {HISTogram}

Returned Format [:MEASure:HISTogram:MEAN] <value>[,<result_state>]<NL>

<value> The mean of the histogram.

<result_state> If SENDvalid is ON, the result state is returned with the measurement result. Refer to the MEASure:RESults command for a list of the result states.

Example The following example returns the mean of the current histogram and prints the result to the controller screen.

```
10 OUTPUT 707;":SYSTEM:HEADER OFF"           !Response headers off
20 OUTPUT 707;":MEASURE:HISTOGRAM:MEAN?"
30 ENTER 707;Histmean
40 PRINT Histmean
50 END
```

HISTogram:MEDian?

Query :MEASURE:HISTogram:MEDian? [<source>]

This query returns the median of the histogram. The median of the histogram is the time or voltage of the point at which 50% of the histogram is to the left or right (above or below for vertical histograms). The source can be specified with the optional parameter following the MEDian query. The HISTogram:MEDian? query only applies to the histogram waveform.

<source> {HISTogram}

Returned Format [:MEASure:HISTogram:MEDian] <value>[,<result_state>]<NL>

<value> The median of the histogram.

<result_state> If SENDvalid is ON, the result state is returned with the measurement result. Refer to the MEASure:RESults command for a list of the result states.

Example The following example returns the median of the current histogram and prints the result to the controller screen.

```
10 OUTPUT 707;":SYSTEM:HEADER OFF"          !Response headers off
20 OUTPUT 707;":MEASURE:HISTOGRAM:MEDIAN?"
30 ENTER 707;Histmed
40 PRINT Histmed
50 END
```

HISTogram:PEAK?

Query :MEASURE:HISTogram:PEAK? [<source>]

This query returns the number of hits in the greatest peak of the histogram. The source can be specified with the optional parameter following the PEAK query. The HISTogram:PEAK? query only applies to the histogram waveform.

<source> {HISTogram}

Returned Format [:MEASure:HISTogram:PEAK] <value>[,<result_state>]<NL>

<value> The number of hits in the histogram peak.

<result_state> If SENDvalid is ON, the result state is returned with the measurement result. Refer to the MEASure:RESults command for a list of the result states.

Example The following example returns the number of hits in the greatest peak of the current histogram and prints the result to the controller screen.

```
10 OUTPUT 707;":SYSTEM:HEADER OFF"          !Response headers off
20 OUTPUT 707;":MEASURE:HISTOGRAM:PEAK?"
30 ENTER 707;Histpeak
40 PRINT Histpeak
50 END
```

HISTogram:PP?

Query :MEASURE:HISTogram:PP? [<source>]

This query returns the width of the histogram. The width is measured as the time or voltage of the last histogram bucket with data in it minus the time or voltage of the first histogram bucket with data in it. The source can be specified with the optional parameter following the PP query. The HISTogram:PP? query only applies to the histogram waveform.

<source> {HISTogram}

Returned Format [:MEASure:HISTogram:PP] <value>[,<result_state>]<NL>

<value> The width of the histogram.

<result_state> If SENDvalid is ON, the result state is returned with the measurement result. Refer to the MEASure:RESults command for a list of the result states.

Example The following example returns the width of the current histogram and prints the result to the controller screen.

```
10 OUTPUT 707;":SYSTEM:HEADER OFF"              !Response headers off
20 OUTPUT 707;":MEASURE:HISTOGRAM:PP?"
30 ENTER 707;Histpp
40 PRINT Histpp
50 END
```

HISTogram:SCALe?

Query :MEASURE:HISTogram:SCALe? [<source>]

The query returns the scale of the histogram in hits per division. The source can be specified with the optional parameter following the SCALe query. The HISTogram:SCALe? query only applies to the histogram waveform.

<source> {HISTogram}

Returned Format [:MEASure:HISTogram:SCALe] <value>[,<result_state>]<NL>

<value> The scale of the histogram in hits.

<result_state> If SENDvalid is ON, the result state is returned with the measurement result. Refer to the MEASure:RESults command for a list of the result states.

Example The following example returns the scale of the histogram whose source is specified in MEASure:SOURce and prints the result to the controller screen.

```
10 OUTPUT 707;":SYSTEM:HEADER OFF"              !Response headers off
20 OUTPUT 707;":MEASURE:HISTOGRAM:SCALE?"
30 ENTER 707;Histscal
40 PRINT Histscal
50 END
```

HISTogram:STDDev?

Query :MEASURE:HISTogram:STDDev? [<source>]

This query returns the standard deviation of the histogram. The source can be specified with the optional parameter following the STDDev query. The HIS-Togram:STDDev? query only applies to the histogram waveform.

<source> {HISTogram}

Returned Format [:MEASure:HISTogram:STDDev] <value>[,<result_state>]<NL>

<value> The standard deviation of the histogram.

<result_state> If SENDvalid is ON, the result state is returned with the measurement result. Refer to the MEASure:RESults command for a list of the result states.

Example The following example returns the standard deviation of the histogram whose source is specified using the MEASure:SOURce command, and prints the result to the controller screen.

```
10 OUTPUT 707;":SYSTEM:HEADER OFF"          !Response headers off
20 OUTPUT 707;":MEASURE:HISTOGRAM:STDDEV?"
30 ENTER 707;Histstdd
40 PRINT Histstdd
50 END
```

NWIDth

Command :MEASure:NWIDth [<source>]

Measures the width of the first negative pulse on the screen using the mid-threshold levels of the waveform (50% levels with standard measurements selected). The source is specified with the MEASure:SOURce command or with the optional parameter following the NWIDth command.

The algorithm is:

If the first edge on screen is rising, then

nwidth = time at the second rising edge – time at the first falling edge

else,

nwidth = time at the first rising edge – time at the first falling edge.

Mode Oscilloscope mode only

<source> {CHANNel<N> | FUNCtion<N> | RESPonse<N> | WMEMory<N>}

<number> For channels: Value is dependent on the type of plug-in and its location in the instrument. For functions: 1 or 2. For waveform memories (WMEMORY): 1, 2, 3, or 4. For TDR responses: 1, 2, 3, or 4.

Example The following example measures the width of the first negative pulse on screen.

```
10 OUTPUT 707;":MEASURE:NWIDTH"
20 END
```

Query :MEASure:NWIDth? [<source>]

The query returns the measured width of the first negative pulse of the specified source.

Returned Format [:MEASure:NWIDth] <value>[,<result_state>]<NL>

<value> The width of the first negative pulse on the screen using the mid-threshold levels of the waveform.

<result_state> If SENDVALID is ON, the result state is returned with the measurement result. Refer to the MEASure:RESults command, for a list of the result states.

Example The following example places the current width of the first negative pulse on screen in the numeric variable, Width, then prints the contents of the variable to the controller's screen.

```
10 OUTPUT 707;":SYSTEM:HEADER OFF" !Response headers off
20 OUTPUT 707;":MEASURE:NWIDTH?"
30 ENTER 707;Width
40 PRINT Width
50 END
```

OVERshoot

Command :MEASure:OVERshoot [<source>]

This command measures the overshoot of the first edge on the screen. Sources are specified with the MEASure:SOURce command or with the optional parameter following the OVERshoot command.

The algorithm is:

If the first edge onscreen is rising, then

overshoot = (Local Vmax - Vtop) / Vamplitude

else

overshoot = (Vbase – Local Vmin) / Vamplitude.

Mode Oscilloscope mode only

<source> {CHANnel<N> | FUNCtion<N> | RESPonse<N> | WMEMory<N>}

<N> For channels, functions, TDR responses and waveform memories: 1, 2, 3, or 4.

Example This example measures the overshoot of the first edge onscreen.

```
10 OUTPUT 707;":MEASURE:OVERSHOOT"
20 END
```

Query :MEASure:OVERshoot? [<source>]

The query returns the measured overshoot of the specified source.

Returned Format [:MEASure:OVERshoot] <value>[,<result_state>]<NL>

<value> Ratio of overshoot to amplitude, in percent.

<result_state> If SENDvalid is ON, the result state is returned with the measurement result. See the MEASure:RESults table in this chapter for a list of the result states.

Example This example places the current value of overshoot in the numeric variable, Value, then prints the contents of the variable to the computer's screen.

```
10 OUTPUT 707;":SYSTEM:HEADER OFF"              !Response headers off
20 OUTPUT 707;":MEASURE:OVERSHOOT?"
30 ENTER 707;Value
40 PRINT Value
50 END
```

PERiod

Command :MEASure:PERiod [<source>]

This command measures the period of the first complete cycle on the screen using the mid-threshold levels of the waveform (50% levels with standard measurements selected). The source is specified with the MEASure:SOURce command or with the optional parameter following the PERiod command.

The algorithm is:

```
If the first edge onscreen is rising then
 period = time at the second rising edge – time at the first rising edge
else
 period = time at the second falling edge – time at the first falling edge.
```

Mode Oscilloscope mode only

<source> {CHANnel<N> | FUNCtion<N> | RESPonse<N> | WMEMory<N>}

<N> For channels, functions, TDR responses and waveform memories: 1, 2, 3, or 4.

Example This example measures the period of the waveform.

```
10 OUTPUT 707;":MEASURE:PERIOD"
20 END
```

Query :MEASure:PERiod? [<source>]

The query returns the measured period of the specified source.

Returned Format [:MEASure:PERiod] <value>[,<result_state>]<NL>

<value> Period of the first complete cycle onscreen.

<result_state> If SENDvalid is ON, the result state is returned with the measurement result. See the MEASure:RESults table in this chapter for a list of the result states.

Example This example places the current period of the waveform in the numeric variable, Value, then prints the contents of the variable to the computer's screen.

```
10 OUTPUT 707;":SYSTEM:HEADER OFF"      !Response headers off
20 OUTPUT 707;":MEASURE:PERIOD?"
30 ENTER 707;Value
40 PRINT Value
50 END
```

PWIDth

Command :MEASure:PWIDth [<source>]

Measures the width of the first positive pulse on the screen using the mid-threshold levels of the waveform (50% levels with standard measurements selected). The source is specified with the MEASure:SOURce command or with the optional parameter following the PWIDth command.

The algorithm is:

If the first edge on screen is rising, then

pwidth = time at the first falling edge – time at the first rising edge

else,

pwidth = time at the second falling edge – time at the first rising edge

Mode Oscilloscope mode only

<source> {CHANnel<N> | FUNCtion<N> | RESPonse<N> | WMEMory<N>}

<N> For channels: Value is dependent on the type of plug-in and its location in the instrument. For functions: 1 or 2. For waveform memories (WMEMORY): 1, 2, 3, or 4. For TDR responses: 1, 2, 3, or 4.

Example The following example measures the width of the first positive pulse on the screen.

```
10 OUTPUT 707;”:MEASURE:PWIDTH”
20 END
```

Query :MEASure:PWIDth? [<source>]

The query returns the measured width of the first positive pulse of the specified source.

Returned Format [:MEASure:PWIDth] <value>[,<result_state>]<NL>

<value> Width of the first positive pulse on the screen in seconds.

<result_state> If SENDVALID is ON, the result state is returned with the measurement result. Refer to the MEASure:RESults command, for a list of the result states.

Example The following example places the value of the width of the first positive pulse on the screen in the numeric variable, Width, then prints the contents of the variable to the controller's screen.

```
10 OUTPUT 707;":SYSTEM:HEADER OFF" !Response headers off
20 OUTPUT 707;":MEASURE:PWIDTH?"
30 ENTER 707;Width
40 PRINT Width
50 END
```

RESults?

Query :MEASure:RESults?

This query returns the results of the continuous measurements. The measurement results always include only the current results. If SENDvalid is ON, the measurement results state is returned immediately following the measurement result. The measurement results include the current, minimum, maximum, mean, standard deviation, and statistical sample size of each measurement.

If more than one measurement is running continuously, the values shown in Table 22-3 on page 22-32 will be duplicated for each continuous measurement from the first to last (top to bottom) of display. There may be up to four continuous measurements at a time.

Returned Format [:MEASure:RESults] <result list><NL>

<result list> A list of the measurement results, as in Table 22-2, separated with commas.

Table 22-2. Results Values

	Sendvalid OFF	Sendvalid ON
Limit test OFF	current result	current result
		validity
	minimum	minimum
	maximum	maximum
	mean	mean
	standard deviation	standard deviation
	n-samples	n-samples
Limit test ON	current result	current result
		validity

Table 22-2. Results Values (Continued)

	Sendvalid OFF	Sendvalid ON
	minimum	minimum
	maximum	maximum
	mean	mean
	standard deviation	standard deviation
	n-samples	n-samples
	limit failures	limit failures
	limit total tests	limit total tests
	limit status	limit status

Example

This example places the current results of the measurements in the string variable, Result$, then prints the contents of the variable to the computer's screen.

```
10 DIM Result$[200]                    !Dimension variable
20 OUTPUT 707;":MEASURE:RESULTS?"
30 ENTER 707;Result$
40 PRINT Result$
50 END
```

Table 22-3. Result States

Code	Result	Description
0	RESULT_CORRECT	Result correct. No problem found.
1	RESULT_QUESTIONABLE	Result questionable but could be measured.
2	RESULT_LESS_EQ	Result less than or equal to value returned.
3	RESULT_GTR_EQ	Result greater than or equal to value returned.
4	RESULT_INVALID	Result returned is invalid.
5	EDGE_NOT_FOUND	Result invalid. Required edge not found.
6	MAX_NOT_FOUND	Result invalid. Max not found.
7	MIN_NOT_FOUND	Result invalid. Min not found.
8	TIME_NOT_FOUND	Result invalid. Requested time not found.
9	VOLT_NOT_FOUND	Result invalid. Requested voltage not found.
10	TOP_EQUALS_BASE	Result invalid. Top and base are equal.
11	MEAS_ZONE_SMALL	Result invalid. Measurement zone too small.
12	LOWER_INVALID	Result invalid. Lower threshold not on waveform.
13	UPPER_INVALID	Result invalid. Upper threshold not on waveform.
14	UPPER_LOWER_INVALID	Result invalid. Upper and lower thresholds are too close.
15	TOP_INVALID	Result invalid. Top not on waveform.
16	BASE_INVALID	Result invalid. Base not on waveform.

Table 22-3. Result States (Continued)

17	INCOMPLETE	Result invalid. Completion criteria not reached.
18	INVALID_SIGNAL	Result invalid. Measurement invalid for this type of signal.
19	SIGNAL_NOT_DISPLAYED	Result invalid. Signal is not displayed.
20	CLIPPED_HIGH	Result invalid. Waveform is clipped high.
21	CLIPPED_LOW	Result invalid. Waveform is clipped low.
22	CLIPPED_HIGH_LOW	Result invalid. Waveform is clipped high and low.
23	ALL_HOLES	Result invalid. Data contains all holes.
24	NO_DATA	Result invalid. No data on screen.
25	CURSOR_OFF_SCREEN	Result invalid. Cursor is not on screen.
26	MEASURE_CANCELLED	Result invalid. Measurement aborted.
27	MEASURE_TIMEOUT	Result invalid. Measurement timed-out.
28	NO_MEAS	Result invalid. No measurement to track.
30	INVALID_EYE	Result invalid. Eye pattern not found.
32	BAD_DARK_LEVEL	Result invalid. Dark level is invalid.
33	NOT_1_SOURCE	Result invalid. Color grade/gray scale database has more than one source.

RISetime

Command :MEASure:RISetime [<source>]

This command measures the rise time of the first displayed edge by measuring the time at the lower threshold of the rising edge, measuring the time at the upper threshold of the rising edge, then calculating the rise time with the following algorithm:

Rise time = time at upper threshold point – time at lower threshold point.

Sources are specified with the MEASure:SOURce command or with the optional parameter following the RISetime command.

Mode Eye and Oscilloscope mode

<source> {CHANnel<N> | FUNCtion<N> | RESPonse<N> | WMEMory<N> | CGRade}

Where CHANnel<N>, FUNCtion<N>, RESPonse<N>, and WMEMory<N> apply in Oscilloscope mode only, and CGRade in Eye mode only.

<N> For channels, functions, TDR responses and waveform memories: 1, 2, 3, or 4.

With standard measurements selected, the lower threshold is at the 10% point and the upper threshold is at the 90% point on the rising edge.

Example This example measures the rise time of the displayed signal.

```
10 OUTPUT 707;":MEASURE:RISETIME"
20 END
```

Query :MEASure:RISetime? [<source>]

The query returns the rise time of the specified source.

Returned Format [:MEASure:RISetime] <value>[,<result_state>]<NL>

<value> Rise time in seconds.

<result_state> If SENDvalid is ON, the result state is returned with the measurement result. See the MEASure:RESults table in this chapter for a list of the result states.

Example This example places the current value of rise time in the numeric variable, Rise, then prints the contents of the variable to the computer's screen.

```
10 OUTPUT 707;":SYSTEM:HEADER OFF"!Response headers off
20 OUTPUT 707;":MEASURE:RISETIME?"
30 ENTER 707;Rise
40 PRINT Rise
50 END
```

SCRatch

Command :MEASure:SCRatch

This command clears the measurement results from the screen.

Example This example clears the current measurement results from the screen.

```
10 OUTPUT 707;":MEASURE:SCRATCH"
20 END
```

SENDvalid

Command :MEASure:SENDvalid {{OFF | 0} | {ON | 1}}

This command enables the result state code to be returned with the :MEASure:RESults? query.

Example This example turns send valid function on.

```
10 OUTPUT 707;":MEASURE:SENDVALID ON"
20 END
```

Query :MEASure:SENDvalid?

The query returns the state of the Sendvalid control.

Returned Format [:MEASure:SENDvalid] {0 | 1}<NL>

Example This example places the current mode for SENDvalid in the string variable, Mode$, then prints the contents of the variable to the computer's screen.

```
10 DIM Mode$[50]                                   !Dimension variable
20 OUTPUT 707;":MEASURE:SENDVALID?"
30 ENTER 707;Mode$
40 PRINT Mode$
50 END
```

See Also Refer to the MEASure:RESults query for information on the results returned and how they are affected by the SENDvalid command. Refer to the individual measurements for information on how the result state is returned.

SOURce

Command :MEASure:SOURce <source>[,<source>]

This command selects the source for measurements. You can specify one or two sources with this command. All measurements except MEASure: DEFine:DELTatime are made on the first specified source. The delta time measurement uses two sources if two are specified. If only one source is specified, the delta time measurement uses that source for both of its parameters. The source is always color grade/gray scale data in eye mode, except for average optical power and histogram measurements.

This is a global definition. It is used for all subsequent remote measurements unless a different source is specified with the optional source parameter in the measure command.

Mode Oscilloscope mode. Eye mode uses this for average optical power measurements.

<source> {CHANNel<N> | FUNCtion<N> | RESPonse<N> | WMEMory<N>}

<N> For channels, functions, TDR responses and waveform memories: 1, 2, 3, or 4.

Example This example selects channel 1 as the source for measurements.

```
10 OUTPUT 707;":MEASURE:SOURCE CHANNEL1"
20 END
```

Query :MEASure:SOURce?

The query returns the current source selection.

Returned Format [:MEASure:SOURce] <source>[,<source>]<NL>

Example This example places the currently specified sources in the string variable, Source$, then prints the contents of the variable to the computer's screen.

```
10 DIM Source$[50]                    !Dimension variable
20 OUTPUT 707;":MEASURE:SOURCE?"
30 ENTER 707;Source$
40 PRINT Source$
50 END
```

TEDGe?

Query :MEASure:TEDGe? <meas_thres_txt>,<slope><occurrence> [,<source>]

The query returns the time interval between the trigger event and the specified edge (threshold level, slope, and transition).

<meas_thres_txt> UPPer, MIDDle, or LOWer to identify the threshold.

<slope> { – (minus) for falling | + (plus) for rising | <none> (the slope is optional; if no slope is specified, + (plus) is assumed) }

<occurrence> A numeric value representing the edge of the occurrence. The desired edge must be present on the display. Edges are counted with 1 being the first edge from the left on the display.

<source> {CHANNel<N> | FUNCtion<N> | RESPonse<N> | WMEMory<N>}

<N> For channels, functions, TDR responses and waveform memories 1, 2, 3, or 4.

Returned Format [:MEASure:TEDGe] <time>[,<result_state>]<NL>

<time> The time interval between the trigger event and the specified voltage level and transition.

<result_state> If SENDvalid is ON, the result state is returned with the measurement result. See the MEASure:RESults table in this chapter for a list of the result states.

Example This example returns the time interval between the trigger event and the 90% threshold on the second rising edge of the source waveform to the numeric variable, Time. The contents of the variable are then printed to the computer's screen.

```
10 OUTPUT 707;":SYSTEM:HEADER OFF"         !Response headers off
20 OUTPUT 707;":MEASURE:TEDGE? UPPER,+2"
30 ENTER 707;Time
40 PRINT Time
50 END
```

Turn Off Headers

When receiving numeric data into numeric variables, turn off the headers. Otherwise, the headers may cause misinterpretation of returned data.

TVOLt?

Query :MEASure:TVOLt? <voltage>,<slope><occurrence>[,<source>]

The query returns the time interval between the trigger event and the specified voltage level and transition. The source is specified with the MEASure:SOURce command or with the optional parameter following the TVOLt? query.

Mode	Oscilloscope mode only
<voltage>	Voltage level at which time will be measured.
<slope>	The direction of the waveform change when the specified voltage is crossed, rising (+) or falling (–).
<occurrence>	The number of the crossing to be reported. If one, the first crossing is reported; if two, the second crossing is reported, and so on.
<source>	{CHANnel<N> | FUNCtion<N> | WMEMory<N>
<N>	For channels, functions, and waveform memories 1, 2, 3, or 4.
Returned Format	[:MEASure:TVOLt] <time>[,<result_state>]<NL>
<time>	The time interval between the trigger event and the specified voltage level and transition.
<result_state>	If SENDVALID is ON, the result state is returned with the measurement result. Refer to the MEASure:RESults command for a list of the result states.

Example The following example returns the time interval between the trigger event and the transition through –.250 Volts on the third rising edge of the source waveform to the numeric variable, Time. The contents of the variable are then printed to the controller's screen.

```
10 OUTPUT 707;":SYSTEM:HEADER OFF" !Response headers off
20 OUTPUT 707;":MEASURE:TVOLT? -.250,+3"
30 ENTER 707;Time
40 PRINT Time
50 END
```

Turn Off Headers

When receiving numeric data into numeric variables, turn off the headers. Otherwise, the headers may cause misinterpretation of returned data.

VAMPlitude

Command :MEASure:VAMPlitude [<source>]

This command calculates the difference between the top and base voltage of the specified source. Sources are specified with the MEASure:SOURce command or with the optional parameter following the VAMPlitude command.

Mode Oscilloscope mode only

<source> {CHANnel<N> | FUNCtion<N> | RESPonse<N> | WMEMory<N>}

<N> For channels, functions, TDR responses and waveform memories: 1, 2, 3, or 4.

Example This example calculates the difference between the top and base voltage of the specified source.

```
10 OUTPUT 707;":MEASURE:VAMPLITUDE"
20 END
```

Query :MEASure:VAMPlitude? [<source>]

The query returns the calculated difference between the top and base voltage of the specified source.

Returned Format [:MEASure:VAMPlitude] <value>[,<result_state>]<NL>

<value> Calculated difference between the top and base voltage.

<result_state> If SENDvalid is ON, the result state is returned with the measurement result. See the MEASure:RESults table in this chapter for a list of the result states.

Example This example places the current Vamplitude value in the numeric variable, Value, then prints the contents of the variable to the computer's screen.

```
10 OUTPUT 707;":SYSTEM:HEADER OFF"          !Response headers off
20 OUTPUT 707;":MEASURE:VAMPLITUDE?"
30 ENTER 707;Value
40 PRINT Value
50 END
```

VBASe

Command	:MEASure:VBASe [<source>]
	Measures the statistical base of the waveform. The source is specified with the MEASure:SOURce command or with the optional parameter following the VBASe command.
Mode	Oscilloscope mode only
<source>	{CHANnel<N> | FUNCtion<N> | RESPonse<N> | WMEMory<N>}
<N>	For channels: Value is dependent on the type of plug-in and its location in the instrument. For functions: 1 or 2. For waveform memories (WMEMORY): 1, 2, 3, or 4. For TDR responses: 1, 2, 3, or 4.
Example	The following example measures the voltage at the base of the waveform.

```
10 OUTPUT 707;":MEASURE:VBASE"
20 END
```

Query	:MEASure:VBASe? [<source>]
	The query returns the measured voltage value at the base of the specified source.
Returned Format	[:MEASure:VBASe] <value>[,<result_state>]<NL>
<value>	Voltage at the base of the waveform.
<result_state>	If SENDVALID is ON, the result state is returned with the measurement result. Refer to the MEASure:RESults command for a list of the result states.
Example	The following example returns the current voltage at the base of the waveform to the numeric variable, Voltage, then prints the contents of the variable to the controller's screen.

```
10 OUTPUT 707;":SYSTEM:HEADER OFF" !Response headers off
20 OUTPUT 707;":MEASURE:VBASE?"
30 ENTER 707;Voltage
40 PRINT Voltage
50 END
```

VMAX

Command	:MEASure:VMAX [<source>]
	Measures the absolute maximum voltage present on the selected source waveform. The source is specified with the MEASure:SOURce command or with the optional parameter following the VMAX command.

Mode Oscilloscope mode only

<source> {CHANnel<N> | FUNCtion<N> | RESPonse<N> | WMEMory<N>}

<N> For channels: Value is dependent on the type of plug-in and its location in the instrument. For functions: 1 or 2. For waveform memories (WMEMORY): 1, 2, 3, or 4. For TDR responses: 1, 2, 3, or 4.

Example The following example measures the absolute maximum voltage on the waveform.

```
10 OUTPUT 707;":MEASURE:VMAX"
20 END
```

Query :MEASure:VMAX? [<source>]

The query returns the measured absolute maximum voltage present on the selected source waveform.

Returned Format [:MEASure:VMAX] <value>[,<result_state>]<NL>

<value> Absolute maximum voltage present on the waveform.

<result_state> If SENDVALID is ON, the result state is returned with the measurement result. Refer to the MEASure:RESults command for a list of the result states.

Example The following example returns the measured absolute maximum voltage on the waveform to the numeric variable, Maximum, then prints the contents of the variable to the controller's screen.

```
10 OUTPUT 707;":SYSTEM:HEADER OFF" !Response headers off
20 OUTPUT 707;":MEASURE:VMAX?"
30 ENTER 707;Maximum
40 PRINT Maximum
50 END
```

VMIN

Command :MEASure:VMIN [<source>]

Measures the absolute minimum voltage present on the selected source waveform. The source is specified with the MEASure:SOURce command or with the optional parameter following the VMIN command.

Mode Oscilloscope mode only

<source> {CHANnel<N> | FUNCtion<N> | RESPonse<N> | WMEMory<N>}

<N> For channels: Value is dependent on the type of plug-in and its location in the instrument. For functions: 1 or 2. For waveform memories (WMEMORY): 1, 2, 3, or 4. For TDR responses: 1, 2, 3, or 4.

Example The following example measures the absolute minimum voltage on the waveform.

```
10 OUTPUT 707;":MEASURE:VMIN"
20 END
```

Query :MEASure:VMIN? [<source>]

The query returns the measured absolute minimum voltage present on the selected source waveform.

Returned Format [:MEASure:VMIN] <value>[,<result_state>]<NL>

<value> Absolute minimum voltage present on the waveform.

<result_state> If SENDVALID is ON, the result state is returned with the measurement result. Refer to the MEASure:RESults command for a list of the result states.

Example The following example returns the measured absolute minimum voltage on the waveform to the numeric variable, Minimum, then prints the contents of the variable to the controller's screen.

```
10 OUTPUT 707;":SYSTEM:HEADER OFF" !Response headers off
20 OUTPUT 707;":MEASURE:VMIN?"
30 ENTER 707;Minimum
40 PRINT Minimum
50 END
```

VPP

Command :MEASure:VPP [<source>]

This command measures the maximum and minimum voltages on the selected source, then calculates the peak-to-peak voltage as the difference between the two voltages. Sources are specified with the MEASure:SOURce command or with the optional parameter following the VPP command.

Mode Oscilloscope mode only

<source> {CHANnel<N> | FUNCtion<N> | RESPonse<N> | WMEMory<N>}

<N> For channels, functions, TDR responses and waveform memories: 1, 2, 3, or 4.

Example This example measures the peak-to-peak voltage.

```
10 OUTPUT 707;":MEASURE:VPP"
20 END
```

Query :MEASure:VPP? [<source>]

The query returns the specified source peak-to-peak voltage.

Returned Format [:MEASure:VPP] <value>[,<result_state>]<NL>

<value> Peak-to-peak voltage of the selected source.

<result_state>	If SENDvalid is ON, the result state is returned with the measurement result. See the MEASure:RESults table in this chapter for a list of the result states.
Example	This example places the current peak-to-peak voltage in the numeric variable, Voltage, then prints the contents of the variable to the computer's screen.

```
10 OUTPUT 707;":SYSTEM:HEADER OFF"      !Response headers off
20 OUTPUT 707;":MEASURE:VPP?"
30 ENTER 707;Voltage
40 PRINT Voltage
50 END
```

VRMS

Command	:MEASure:VRMS {CYCLe | DISPlay}, {AC | DC} [,<source>]
	This command measures the RMS voltage of the selected waveform by subtracting the average value of the waveform from each data point on the display. Sources are specified with the MEASure:SOURce command or with the optional parameter following the VRMS command.
Mode	Oscilloscope mode only
CYCLe	The CYCLe parameter instructs the RMS measurement to measure the RMS voltage across the first period of the display.
DISPlay	The DISPLay parameter instructs the RMS measurement to measure all the data on the display. Generally, RMS voltage is measured across one waveform or cycle, however, measuring multiple cycles may be accomplished with the DISPLay option. The DISPlay parameter is also useful when measuring noise.
AC	The AC parameter is used to measure the RMS voltage subtracting out the DC component.
DC	The DC parameter is used to measure RMS voltage including the DC component.
	The AC RMS, DC RMS, and VAVG parameters are related as in the following formula:

$$DCVRMS^2 = ACVRMS^2 + VAVG^2$$

<source>	{CHANnel<N> | FUNCtion<N> | RESPonse<N> | WMEMory<N>}
<N>	For channels, functions, TDR responses and waveform memories: 1, 2, 3, or 4.
Example	This example measures the RMS voltage of the previously selected waveform.

```
10 OUTPUT 707;":MEASURE:VRMS CYCLE,AC"
20 END
```

Query :MEASure:VRMS? {CYCLe | DISplay}, {AC | DC} [,<source>]

The query returns the RMS voltage of the specified source.

Returned Format [:MEASure:VRMS] <value>[,<result_state>]<NL>

<value> RMS voltage of the selected waveform.

<result_state> If SENDvalid is ON, the result state is returned with the measurement result. See the MEASure:RESults table in this chapter for a list of the result states.

Example This example places the current AC RMS voltage over one period of the waveform in the numeric variable, Voltage, then prints the contents of the variable to the computer's screen.

```
10 OUTPUT 707;":SYSTEM:HEADER OFF"      !Response headers off
20 OUTPUT 707;":MEASURE:VRMS? CYCLE,AC"
30 ENTER 707;Voltage
40 PRINT Voltage
50 END
```

VTIMe?

Query :MEASure:VTIMe? <time> [,<source>]

The query returns the measured voltage.

Mode Oscilloscope mode only

Returned Format [:MEASure:VTIMe] <value>[,<result_state>]<NL>

<value> Voltage at the specified time.

<result_state> If SENDVALID is ON, the result state is returned with the measurement result. Refer to the MEASure:RESults command for a list of the result states.

Example The following example places the voltage at 500 ms in the numeric variable, Value, then prints the contents to the controller's screen.

```
10 OUTPUT 707;":SYSTEM:HEADER OFF" !Response headers off
20 OUTPUT 707;":MEASURE:VTIME? 500E–3"
30 ENTER 707;Value
40 PRINT Value
50 END
```

VTOP

Command	:MEASure:VTOP [<source>]
	This command measures the statistical top of the selected source waveform. The source is specified with the MEASure:SOURce command or with the optional parameter following the VTOP command.
Mode	Oscilloscope mode only
<source>	{CHANnel<N> | FUNCtion<N> | RESPonse<N> | WMEMory<N>}
<N>	For channels: Value is dependent on the type of plug-in and its location in the instrument. For functions: 1 or 2. For waveform memories (WMEMORY): 1, 2, 3, or 4. For TDR responses: 1, 2, 3, or 4.
Example	The following example measures the voltage at the top of the waveform.

```
10 OUTPUT 707;":MEASURE:VTOP"
20 END
```

Query	:MEASure:VTOP? [<source>]
	The query returns the measured voltage at the top of the specified source.
Returned Format	[:MEASure:VTOP] <value>[,<result_state>]<NL>
<value>	Voltage at the top of the waveform.
<result_state>	If SENDVALID is ON, the result state is returned with the measurement result. Refer to the MEASure:RESults command for a list of the result states.
Example	The following example places the value of the voltage at the top of the waveform in the numeric variable, Value, then prints the contents of the variable to the controller's screen.

```
10 OUTPUT 707;":SYSTEM:HEADER OFF" !Response headers off
20 OUTPUT 707;":MEASURE:VTOP?"
30 ENTER 707;Value
40 PRINT Value
50 END
```

23

TDR/TDT Commands

TDR/TDT Commands

The TDR/TDT command subsystem includes all commands necessary to set up TDR/TDT measurements.

Slot Selection

All of the TDR/TDT subsystem commands are of the form :TDR{2 | 4}:<command>. The {2 | 4} option is used to identify the slot in which you have installed the TDR/TDT plug-in module. Select 2 if the module is in slots 1 and 2; 4 if the module is in slots 3 and 4. For example, if the module is in slots 3 and 4, and you want to issue the TDR subsystem PRESet command, you use the command string :TDR4:PRESET.

PRESet

This command is used in TDR/TDT mode only.

Command

```
:TDR{2 | 4}:PRESet
```

This command performs an automatic set up of the instrument for TDR or TDT measurements, based on the stimulus. This command does the following:

- Turn on TDR channels.
- If the TDT destinations are not shown, turn on the TDT destination channels. (See the TDTDest command in this chapter).
- Set the timebase to 500 ps/div and positions the incident edge on screen.
- Turn on averaging and set best flatness (see the ACQuire subsystem).
- For all channels that are on:
  - Set the attenuation units to ratio.
  - Set the attenuation to 1:1.
  - Set the bandwidth to low (12.4 GHz). (Set high for external stimulus.)
  - Set the units to volts.
  - Set the channel scale to 100 mV/div.
  - Set the channel offset to 200 mV or –200 mV for differential stimulus.

Example

The following example presets the instrument for TDR/TDT operations.

```
10 OUTPUT 707;":TDR2:PRESET"
20 END
```

RATE

This command is used in TDR/TDT mode only.

Command

```
:TDR{2 | 4}:RATE {AUTO | <rate>}
```

This command sets the period of the TDR pulse generator. You should usually leave this set to AUTO unless you need to define a specific rate. In AUTO, the instrument will attempt to keep subsequent periods off screen when the time-base is changed.

<rate>

Period to which you want to set the generator, in Hertz. You can add a suffix to indicate that the rate is in Hertz (HZ, KHZ, and so on).

Example The following example sets the pulse generator to 120 kHz.

```
10 OUTPUT 707;":TDR2:RATE 120 KHZ"
20 END
```

Query :TDR{2 | 4}:RATE?

The query returns the current period of the pulse generator, even when the control is set to AUTO.

The query is allowed in all modes.

Returned Format [:TDR{2 | 4}:RATE] {AUTO | <rate>}<NL>

Example The following example gets the current rate setting and stores it in the variable Rate$, then prints the contents of the variable to the controller's screen.

```
10 DIM Rate$[30]
20 OUTPUT 707;":TDR2:RATE?"
30 ENTER 707;Rate$
40 PRINT Rate$
50 END
```

RESPonse

This command is used in TDR/TDT mode only.

Command :TDR{2 | 4}:RESPonse{1 | 2| 3| 4} {OFF | ON | DIFFerential | COMMonmode}

This command turns on or off a TDR or TDT normalized response.

The value following RESPonse (1, 2, 3, 4) specifies the stimulus channel used to produce a response waveform. The numeric value in the RESPonse remote commands is not the same as the number of the response waveform. Response waveforms are numbered based on the destination channel.

OFF Turns off the response for the specified stimulus.

ON Turns on the normalized response of the channel.

Note

The keyword NORMalize may also be used. This command is compatible with the Agilent 83480/54750 and is equivalent to ON.

DIFFerential Turns on the differential response.

COMMonmode Turns on the common mode response.

Example The following example turns on common mode response on response 1.

```
10 OUTPUT 707;":TDR2:RESPONSE1 COMMONMODE"
20 END
```

Query :TDR{2 | 4}:RESPonse{1 | 2 | 3 | 4}?

The query returns the current response setting for the specified stimulus.

The query is allowed in all modes.

Returned Format [:TDR{2 | 4}:RESPonse{1 | 2 | 3 | 4}] {OFF | DIFFerential | COMMonmode | ON}<NL>

Example The following example gets the current response setting for response 2, stores it in the variable Control$, then prints the contents of the variable to the controller's screen.

```
10 DIM Control$[20]
20 OUTPUT 707;":TDR2:RESPONSE2?"
30 ENTER 707;Control
40 PRINT Control
50 END
```

RESPonse:CALibrate

This command is used in TDR/TDT mode only.

Command :TDR{2 | 4}:RESPonse{1 | 2 | 3 | 4}:CALibrate

This command begins a TDR or TDT normalization and reference plane calibration. Which calibration is done (TDR or TDT) depends on the setting of the TDRTDT control. See “RESPonse:TDRTDT” on page 23-11.

The value following RESPonse (1, 2, 3, 4) specifies the stimulus channel used to produce a response waveform. The numeric value in the RESPonse remote commands is not the same as the number of the response waveform. Response waveforms are numbered based on the destination channel.

Example The following example begins a TDR or TDT calibration.

```
10 OUTPUT 707;":TDR2:RESPONSE1:CALIBRATE"
20 END
```

RESPonse:CALibrate:CANCel

This command is used in TDR/TDT mode only.

Command :TDR{2 | 4}:RESPonse{1 | 2 | 3 | 4}:CALibrate:CANCel

This command activates the cancel softkey during a TDR or TDT normalization and reference plane calibration.

This command is retained for backward compatibility with the 83480/54750. The preferred command is :CALibrate:CANCel.

The value following RESPonse (1, 2, 3, 4) specifies the stimulus channel used to produce a response waveform. The numeric value in the RESPonse remote commands is not the same as the number of the response waveform. Response waveforms are numbered based on the destination channel.

Example The following example cancels the current calibration operation.

```
10 OUTPUT 707;":TDR2:RESPONSE1:CALIBRATE:CANCEL"
20 END
```

RESPonse:CALibrate:CONTinue

This command is used in TDR/TDT mode only.

Command :TDR{2 | 4}:RESPonse{1 | 2 | 3 | 4}:CALibrate:CONTinue

This command activates the continue softkey during a TDR or TDT normalization and reference plane calibration.

This command is retained for backward compatibility with the 83480/54750. The preferred command is :CALibrate:CONTinue.

The value following RESPonse (1, 2, 3, 4) specifies the stimulus channel used to produce a response waveform. The numeric value in the RESPonse remote commands is not the same as the number of the response waveform. Response waveforms are numbered based on the destination channel.

Example The following example continues a paused calibration operation.

```
10 OUTPUT 707;":TDR2:RESPONSE1:CALIBRATE:CONTINUE"
20 END
```

RESPonse:HORizontal

This command is used in TDR/TDT mode only.

Command :TDR{2 | 4}:RESPonse{1 | 2 | 3 | 4}:HORizontal {AUTO | MANual}

This command specifies whether the TDR/TDT response should automatically track the source channel’s horizontal scale (AUTO), or a user-defined scale specified with the HORizontal:POSItion and HORizontal:RANGe commands (MANual). AUTO is the usual setting.

Note

The keyword TSOurce may also be used. This command is compatible with the Agilent 83480/54750 and is equivalent to AUTO.

The value following RESPonse (1, 2, 3, 4) specifies the stimulus channel used to produce a response waveform. The numeric value in the RESPonse remote commands is not the same as the number of the response waveform. Response waveforms are numbered based on the destination channel.

Example The following example sets TDR response 1 to automatically track the source channel’s horizontal scale:

```
10 OUTPUT 707;":TDR2:RESPONSE1:HORIZONTAL AUTO"
20 END
```

Query :TDR{2 | 4}:RESPonse{1 | 2 | 3 | 4}:HORizontal?

The query returns the current horizontal tracking mode for the specified response.

Returned Format [:TDR{2 | 4}:RESPonse{1 | 2 | 3 | 4}:HORizontal] {AUTO | MANual}<NL>

Example The following example gets the current horizontal tracking mode for response 1, puts it in the variable Track$, then prints the contents of the variable to the controller’s screen:

```
10 DIM Track$[20]
20 OUTPUT 707;":TDR2:RESPONSE1:HORIZONTAL?"
30 ENTER 707;Track$
40 PRINT Track$
50 END
```

RESPonse:HORizontal:POSition

This command is used in TDR/TDT mode only.

Command :TDR{2 | 4}:RESPonse{1 | 2 | 3 | 4):HORizontal:POSition <position>

This command specifies the horizontal position of the TDR/TDT response when horizontal tracking is set to manual. The position is always referenced to center screen.

The value following RESPonse (1, 2, 3, 4) specifies the stimulus channel used to produce a response waveform. The numeric value in the RESPonse remote commands is not the same as the number of the response waveform. Response waveforms are numbered based on the destination channel.

<position> Offset from the center of the screen, in seconds.

Example The following example sets the horizontal position for response 1 to 20 ns. This assumes that manual tracking has already been selected.

```
10 OUTPUT 707;":TDR2:RESPONSE1:HORIZONTAL:POSITION 20E9"
20 END
```

Query :TDR{2 | 4}:RESPonse{1 | 2 | 3 | 4}:HORizontal:POSition?

The query returns the current horizontal position setting for the specified response.

Returned Format [:TDR{2 | 4}:RESPonse{1 | 2 | 3 | 4):HORizontal:POSition] <position><NL>

Example The following example gets the current horizontal position setting for response 1, puts it into the variable Pos$, then prints the contents of the variable to the controller's screen.

```
10 DIM Pos$[20]
20 OUTPUT 707;":TDR2:RESPONSE1:HORIZONTAL:POSITION?"
30 ENTER 707;Pos$
40 PRINT Pos$
50 END
```

RESPonse:HORizontal:RANGe

This command is used in TDR/TDT mode only.

Command :TDR{2 | 4}:RESPonse{1 | 2 | 3 | 4}:HORizontal:RANGe <range>

This command specifies the range of the TDR/TDT response when the horizontal tracking is set to manual.

The value following RESPonse (1, 2, 3, 4) specifies the stimulus channel used to produce a response waveform. The numeric value in the RESPonse remote commands is not the same as the number of the response waveform. Response waveforms are numbered based on the destination channel.

<range> Horizontal range in seconds.

Example The following example sets the horizontal range for TDR response 1 to 120 ms. This assumes that manual tracking has already been selected.

```
10 OUTPUT 707;":TDR2:RESPONSE1:HORIZONTAL:RANGE 120 MS"
20 END
```

Query :TDR{2 | 4}:RESPonse{1 | 2 | 3 | 4}:HORizontal:RANGe?

The query returns the current horizontal range setting for the specified response.

Returned Format [:TDR{2 | 4}:RESPonse{1 | 2 | 3 | 4}:HORizontal:RANGe] <range><NL>

Example

The following example gets the current horizontal range setting for response 2, stores it in the numeric variable Range, then prints the contents of the variable to the controller's screen.

```
10 OUTPUT 707;":TDR2:RESPONSE2:HORIZONTAL:RANGE?"
20 ENTER 707;Range
30 PRINT Range
40 END
```

RESPonse:RISetime

This command is used in TDR/TDT mode only.

Command

:TDR{2 | 4}:RESPonse{1 | 2 | 3 | 4}:RISetime <risetime>

This command sets the risetime for the normalized response. The risetime setting is limited by the timebase settings and the record length. The normalize response function allows you to change the risetime of the normalized step.

The value following RESPonse (1, 2, 3, 4) specifies the stimulus channel used to produce a response waveform. The numeric value in the RESPonse remote commands is not the same as the number of the response waveform. Response waveforms are numbered based on the destination channel.

<risetime>

Risetime setting in seconds. The Risetime function allows you to change the normalized step's risetime from a minimum of:

10 ps or whichever is greater, to a maximum of max = 5 x time per division (s/div)

While the TDR step's risetime applied to the system under test is fixed, the measured response has a set of mathematical operations applied to it. These mathematical operations effectively change the displayed response to the system just as if a different TDR step risetime had actually been applied. This allows you to select a risetime for TDR/TDT measurements that is close to the actual risetime used in your system. This risetime value applies to both TDR and TDT normalized channels.

Example

The following example sets the risetime for response 1 to 100 ps.

```
10 OUTPUT 707;"TDR2:RESPONSE1:RISETIME 100 PS"
20 END
```

Query

:TDR{2 | 4}:RESPonse{1 | 2 | 3 | 4}:RISetime?

The query returns the normalized response risetime setting.

Returned Format

[:TDR{2 | 4}:RESPonse{1 | 2 | 3 | 4}:RISetime] <risetime><NL>

Example The following example gets the current risetime setting and stores it in the numeric variable Risetime, then prints the contents of the variable to the controller's screen.

```
10 OUTPUT 707;":TDR2:RESPONSE1:RISETIME?"
20 ENTER 707;Risetime
30 PRINT Risetime
40 END
```

RESPonse:TDRDest

This command is used in TDR/TDT mode only.

Command :TDR{2 | 4}:RESPonse{1 | 3}:TDRDest CHANnel<number>

This command selects a TDR destination channel for an external stimulus. When you use an external stimulus, you must use this command to specify where the TDR channel is coming into the instrument. An external stimulus may be generated from channels 1 or 3 only.

A channel is valid as a TDR destination if it meets the following criteria:

- Must be an electrical channel.
- Must not have an active TDR stimulus.
- Must not be the destination of a TDT measurement.

<number> An integer, 1 through 4, indicating the slot in which the channel resides.

Example The following example sets channel 2 as the TDR destination channel for response 1:

```
10 OUTPUT 707;":TDR2:RESPONSE1:TDRDEST CHANNEL2"
20 END
```

Query :TDR{2 | 4}:RESPonse{1 | 3}:TDRDest?

The query returns the current TDR destination channel for the selected response.

Returned Format [:TDR{2 | 4}:RESPonse{1 | 3}:TDRDest] <channel><NL>

Example The following example gets the current TDR destination channel for response 3, stores it in the variable Dest$, then prints the contents of the variable to the controller's screen:

```
10 DIM Dest$[20]
20 OUTPUT 707;":TDR2:RESPONSE3:TDRDEST?"
30 ENTER 707;Dest$
40 PRINT Dest$
50 END
```

RESPonse:TDRTDT

This command is used in TDR/TDT mode only.

Command :TDR{2 | 4}:RESPonse{1| 2| 3 | 4}:TDRTDT {TDR | TDT}

This command controls the behavior of other :TDR{2| 4:RESPonse commands and queries. A response waveform is fully specified by the TDRTDT setting, as well as by the stimulus value that is part of a “TDR{2 | 4}:RESPonse command.

The value following RESPonse (1, 2, 3, 4) specifies the stimulus channel used to produce a response waveform. The numeric value in the RESPonse remote commands is not the same as the number of the response waveform. Response waveforms are numbered based on the destination channel.

Example To turn on Response 1 waveform as TDR with stimulus = Chan1:

Set :TDR2:RESPonse1:TDRTDT to TDR
Set :TDR2:RESPonse1 to NORM

To turn on Response 2 waveform as TDT with stimulus = Chan1:

Set :TDR2:RESPonse:TDTDest to Chan2
Set :TDR2:RESPonse:TDRTDT to TDT
Set :TDR2:RESPonse1 to NORM

RESPonse:TDTDest

This command is used in TDR/TDT mode only.

Command :TDR{2 | 4}:RESPonse{1 | 2 | 3 | 4}:TDTDest {NONE | CHANnel<number>}

This command selects a destination channel for a normalization measurement.

The value following RESPonse (1, 2, 3, 4) specifies the stimulus channel used to produce a response waveform. The numeric value in the RESPonse remote commands is not the same as the number of the response waveform. Response waveforms are numbered based on the destination channel.

For differential and common mode stimuli, the TDT destination is implied as follows:

- The TDT destination for channel 1 is channel 3.
- The TDT destination for channel 2 is channel 4.
- The TDT destination for channel 3 is channel 1.
- The TDT destination for channel 4 is channel 2.

A channel is valid as a TDT destination if it meets the following criteria:

- Must be an electrical channel.
- Must not have an active TDR stimulus.
- Must not be the destination of another TDT measurement.
- Must not be the destination of a TDR measurement (external stimulus only).

You must select a valid TDT destination before setting the TDRTDT control to TDT.

NONE Deselects a channel as a TDT destination. This frees the channel to be the TDT destination of another TDR source.

<number> An integer, 1 through 4, indicating the slot in which the channel resides, followed by an optional A or B identifying which of two possible channels in the slot is being referenced.

Example The following example selects channel 3 as the TDT destination channel for response 4.

```
10 OUTPUT 707;":TDR2:RESPONSE4:TDTDEST CHANNEL3"
20 END
```

Query :TDR{2 | 4}:RESPonse{1 | 2 | 3 | 4}:TDTDest?

The query returns the current TDT destination channel for the specified response.

Returned Format [:TDR{2 | 4}:RESPonse{1 | 2 | 3 | 4}:TDTDest] {NONE | <channel>}<NL>

Example The following example gets the TDT destination channel for response 1, puts it in the variable Dest$, then prints the contents of the variable to the controller's screen.

```
10 DIM Dest$[20]
20 OUTPUT 707;":TDR2:RESPONSE1:TDTDEST?"
30 ENTER 707;Dest$
40 PRINT Dest$
50 END
```

RESPonse:VERTical

This command is used in TDR/TDT mode only.

Command :TDR{2 | 4}:RESPonse{1 | 2 | 3 | 4}:VERTical {AUTO | MANual}

This command specifies whether the TDR/TDT response should automatically track the source channel's vertical scale (AUTO), or use a user-defined scale specified with the VERTical:OFFSet and VERTical:RANGe commands (MANual). AUTO is the usual setting.

> **Note**
>
> The keyword TSOurce may also be used. This command is compatible with the Agilent 83480/54750 and is equivalent to AUTO.

The value following RESPonse (1, 2, 3, 4) specifies the stimulus channel used to produce a response waveform. The numeric value in the RESPonse remote commands is not the same as the number of the response waveform. Response waveforms are numbered based on the destination channel.

Example The following example sets response 1 to use a user-defined vertical scale.

```
10 OUTPUT 707;":TDR2:RESPONSE1:VERTICAL MANUAL"
20 END
```

Query :TDR{2 | 4}:RESPonse{1 | 2 | 3 | 4}:VERTical?

The query returns the current vertical tracking mode for the specified response.

Returned Format [:TDR{2 | 4}:RESPonse{1 | 2 | 3 | 4}:VERTical] {AUTO | MANual}<NL>

Example The following example gets the current vertical tracking mode for response 4, puts it in the variable VertMode$, then prints the contents of the variable to the controller’s screen.

```
10 DIM VertMode$[20]
20 OUTPUT 707;":TDR2:RESPONSE4:VERTICAL?"
30 ENTER 707;VertMode$
40 PRINT VertMode
50 END
```

RESPonse:VERTical:OFFSet

This command is used in TDR/TDT mode only.

Command :TDR{2 | 4}:RESPonse{1 | 2 | 3 | 4}: VERTical:OFFSet <offset_value>

This command sets the vertical position of the specified response when vertical tracking is set to MANual. The position is always referenced to center screen.

The value following RESPonse (1, 2, 3, 4) specifies the stimulus channel used to produce a response waveform. The numeric value in the RESPonse remote commands is not the same as the number of the response waveform. Response waveforms are numbered based on the destination channel.

<offset_value> Offset value in volts, watts, or decibels, depending on the current channel UNITs. Suffix UNITs are ignored; only the scalar part is used (m in mw).

Example The following example sets the vertical offset to 50 mV for response 1. This assumes that the vertical tracking mode has already been set to MANual.

```
10 OUTPUT 707;":TDR2:RESPONSE1:OFFSET 50 MV"
20 END
```

Query :TDR{2 | 4}:RESPonse{1 | 2 | 3 | 4}:VERTical:OFFSet?

The query returns the vertical offset for the specified response. This information is valid only when the vertical tracking mode is set to manual for the response.

Returned Format [:TDR{2 | 4}:RESPonse{1 | 2 | 3 | 4}:VERTical:OFFSet] <volts><NL>

Example The following example gets the vertical offset for response 1, stores it in the numeric variable Offset, then prints the contents of the variable to the controller's screen.

```
10 OUTPUT 707;":TDR2:RESPONSE1:VERTICAL:OFFSET?"
20 ENTER 707;Offset
30 PRINT OFFSET
40 END
```

RESPonse:VERTical:RANGe

This command is used in TDR/TDT mode only.

Command :TDR{2 | 4}:RESPonse{1 | 2 | 3 | 4}:VERTical:RANGe <range_value>

This command specifies the vertical range of the TDR/TDT response when the vertical tracking mode is set to MANual.

The value following RESPonse (1, 2, 3, 4) specifies the stimulus channel used to produce a response waveform. The numeric value in the RESPonse remote commands is not the same as the number of the response waveform. Response waveforms are numbered based on the destination channel.

<range_value> Vertical range in volts, watts, or decibels, depending on the current UNITs setting and suffix supplied. (The suffix does not set the UNITs; it is ignored.)

Example The following example sets the vertical range to 5 volts for response 1. This assumes that the vertical tracking mode has already been set to manual.

```
10 OUTPUT 707;":TDR2:RESPONSE1:VERTICAL:RANGE 5 V"
20 END
```

Query :TDR{2 | 4}:RESPonse{1 | 2 | 3 | 4}:VERTical:RANGe?

The query returns the current vertical range setting for the specified response. This information is valid only when the vertical tracking mode is set to manual.

Returned Format [:TDR{2 | 4}:RESPonse{1 | 2 | 3 | 4}:VERTical:RANGe] <volts><NL>

Example The following example gets the vertical range setting for response 1, stores it in the numeric variable Range, then prints the contents of the variable on the controller's screen.

```
10 OUTPUT 707;":TDR2:RESPONSE1:VERTICAL:RANGE?"
20 ENTER 707;Range
30 PRINT Range
40 END
```

STIMulus

This command is used in TDR/TDT mode only.

Command :TDR{2 | 4}:STIMulus {OFF | ON | ON1 | ON2 | ON1AND2 | DIFFerential | COMMonmode | EXTernal | ON3 | ON4 | ON3AND4}

This command turns the TDR/TDT stimulus on or off. This command is set before starting normalization to specify type of normalization or reference plane calibration to perform.

- The stimulus may be OFF, ON, or EXTernal.
- In slots 1 and 2, the stimulus may be OFF, ON1, ON2, ON1AND2, DIFFerential, or COMMonmode.
- In slots 3 and 4, the stimulus may be OFF, ON3, ON4, ON3AND4, DIFFerential, or COMMonmode.

OFF	Turn off the pulse generator, using the channel as a regular analyzer channel.
ON, ON1, ON3	Turn on the channel 1 or channel 3 pulse generator for single-ended TDR or TDT measurements.
ON2, ON4	Turn on the channel 2 or channel 4 pulse generator for single-ended TDR or TDT measurements.
ON1AND2, ON3AND4	Turn on the pulse generator for channels 1 and 2 or channels 3 and 4 for simultaneous single-ended TDR or TDT measurements.
DIFFerential	Turn on the pulse generator for channels 1 and 2 or channels 3 and 4 for differential TDR or TDT measurements.
COMMonmode	Turn on the pulse generator for channels 1 and 2 or channels 3 and 4 for common-mode TDR or TDT measurements.

Example The following example turns on pulse generators for channels 3 and 4 for single-ended TDR measurements.

```
10 OUTPUT 707;":TDR4:STIMULUS ON3AND4"
20 END
```

Query

:TDR{2 | 4}:STIMulus?

The query returns the current settings for the TDR pulse generators.

Returned Format

[:TDR{2 | 4}:STIMulus] {OFF | ON | ON1 | ON2 | ON1AND2 | DIFFerential | COMMonmode | EXTernal | ON3 | ON4 | ON3AND4}<NL>

Example

The following example gets the current settings of the pulse generators and stores it in the variable Stim$, then prints the contents of that variable to the controller’s screen.

```
10 DIM Stim$[30]
20 OUTPUT 707;":TDR4:STIMULUS?"
30 ENTER 707;Stim$
40 PRINT Stim$
50 END
```

24

Time Base Commands

Time Base Commands

The TIMebase subsystem commands control the horizontal (X axis) analyzer functions.

BRATe

Command :TIMebase:BRATe <bit_rate>

This command sets the bit rate used when the time base units are bit period.

<bit_rate> The bit rate (in bits-per-second).

Example The following example sets the bit rate to 155.520 MHz.

```
10 OUTPUT 707;":TIMEBASE:BRATe 155.520E6"
20 END
```

Query :TIMebase:BRATe?

The query returns the bit rate setting.

Returned Format [:TIMebase:BRATe] <bit_rate><NL>

Example The following example places the current bit rate in the numeric variable, Setting, then prints the contents of the variable to the controller's screen.

```
10 OUTPUT 707;":SYSTEM:HEADER OFF"      !Response headers off
20 OUTPUT 707;":TIMEBASE:BRATe?"
30 ENTER 707;Setting
40 PRINT Setting
50 END
```

POSition

Command :TIMebase:POSition <position_value>

This command sets the time interval between the trigger event and the delay reference point. The delay reference point is set with the TIMebase:REFerence command.

<position_value> The maximum value depends on the time/division setting.

Example This example sets the delay position to 2 ms.

```
10 OUTPUT 707;":TIMEBASE:POSITION 2E-3"
20 END
```

Query :TIMebase:POSition? [{BITS | TIME}]

The query returns the current delay value in seconds.

BITS bits/screen at bit rate

TIME seconds/division

Returned Format [:TIMebase:POSition] <position_value><NL>

Example This example places the current delay value in the numeric variable, Value, then prints the contents of the variable to the computer's screen.

```
10 OUTPUT 707;":SYSTEM:HEADER OFF"      !Response headers off
20 OUTPUT 707;":TIMEBASE:POSITION?"
30 ENTER 707;Value
40 PRINT Value
50 END
```

RANGe

Command :TIMebase:RANGe <full_scale_range>

This command sets the full-scale horizontal time in seconds. The range value is ten times the time-per-division value. Range is always set in units of time (seconds), not in bits.

<full_scale_range> 100 ps to 10 s

Example This example sets the full-scale horizontal range to 10 ms.

```
10 OUTPUT 707;":TIMEBASE:RANGE 10E-3"
20 END
```

Query :TIMebase:RANGe?

The query returns the current full-scale horizontal time.

Returned Format [:TIMebase:RANGe] <full_scale_range><NL>

Example This example places the current full-scale horizontal range value in the numeric variable, Setting, then prints the contents of the variable to the computer's screen.

```
10 OUTPUT 707;":SYSTEM:HEADER OFF"      !Response headers off
20 OUTPUT 707;":TIMEBASE:RANGE?"
30 ENTER 707;Setting
40 PRINT Setting
50 END
```

REFerence

Command :TIMebase:REFerence {LEFT | CENTer}

This command sets the delay reference to the left or center side of the display.

Example This example sets the delay reference to the center of the display.

```
10 OUTPUT 707;":TIMEBASE:REFERENCE CENTER"
20 END
```

Query :TIMebase:REFerence?

The query returns the current delay reference position.

Returned Format [:TIMebase:REFerence] {LEFT | CENTer}<NL>

Example This example places the current delay reference position in the string variable, Setting$, then prints the contents of the variable to the computer's screen.

```
10 DIM Setting$[50]                 !Dimension variable
20 OUTPUT 707;":TIMEBASE:REFERENCE?"
30 ENTER 707;Setting$
40 PRINT Setting$
50 END
```

SCALe

Command :TIMebase:SCALe <value>

This command sets the time base scale. This corresponds to the horizontal scale value displayed as time/div on the analyzer screen.

<value> Value can optionally have units of seconds or bits. If no units are specified <value> has units of the current units setting.

seconds:	time per division
bits:	bits on screen at bit rate setting

Example This example sets the scale to 10 ms/div.

```
10 OUTPUT 707;":TIMEBASE:SCALE 10E-3"
20 END
```

Query :TIMebase:SCALe? [{BITS | TIME}]

The query returns the current scale time setting. If the optional parameter is omitted, the scale value returned is in the units of the current units setting (bits or time).

BITS bits/screen at bit rate

TIME seconds/division

Returned Format [:TIMebase:SCALe] <time><NL>

Example This example places the current scale value in the numeric variable, Setting, then prints the contents of the variable to the computer's screen.

```
10 OUTPUT 707;":SYSTEM:HEADER OFF"!Response headers off
20 OUTPUT 707;":TIMEBASE:SCALE?"
30 ENTER 707;Setting
40 PRINT Setting
50 END
```

UNITs

Command :TIMebase:UNITs {TIME | BITS}

This command sets the time base units.

Example The following example sets the time base units to bits.

```
10 OUTPUT 707;":TIMEBASE:UNITs BITS"
20 END
```

Query :TIMebase:UNITs?

The query returns the time base units.

Returned Format [:TIMebase:UNITs] {TIME | BITS}<NL>

Example The following example places the current bit rate in the numeric variable, Setting, then prints the contents of the variable to the controller's screen.

```
10 OUTPUT 707;":SYSTEM:HEADER OFF"      !Response headers off
20 OUTPUT 707;":TIMEBASE:UNITs?"
30 ENTER 707;Setting
40 PRINT Setting
50 END
```

Trigger Commands

Trigger Commands

The scope trigger circuitry helps you locate the waveform you want to view. Edge triggering identifies a trigger condition by looking for the slope (rising or falling) and voltage level (trigger level) on the source you select. Any input channel, auxiliary input trigger (4-channel scopes only), line, or external trigger (2-channel scopes only) inputs can be used as the trigger source.

The commands in the TRIGger subsystem define the conditions for triggering. The command set has been defined to closely represent the front-panel trigger dialogs.

ATTenuation

Command :TRIGger:ATTenuation <attenuation factor>[,{RATio | DECibel}]

This command controls the attenuation factor and units. The default attenuation factor value is 1:1. The default attenuation units is ratio.

Query :TRIGger:ATTenuation?

The query returns the current attenuation factor and units.

Returned Format [:TRIGger:ATTenuation] <attenuation factor>[,{RATio | DECibel}]<NL>

BWLimit

Command :TRIGger:BWLimit {DIVided | HIGH | LOW}

This command controls an internal lowpass filter and a divider in the 86100A trigger. The bandwidth of the trigger is limited to approximately 100 MHz. DIVided mode is unaffected by the level, hysteresis, and slope settings. The DIVided parameter is only valid if the mainframe has option 001.

Example The following example turns on the bandwidth limit filter for the 86100A trigger:

```
10 OUTPUT 707;":TRIGGER:BWLIMIT LOW"
20 END
```

Query :TRIGger:BWLimit?

The query returns the current setting for the specified trigger input.

Returned Format [:TRIGger:BWLimit] {HIGH | LOW| DIV}<NL>

GATed

Command :TRIGger:GATed {ON | 1 | OFF | 0}

This command enables or disables the ability of the instrument to respond to trigger inputs.

Query :TRIGger:GATed?

The query returns the current gated setting.

Returned Format [:TRIGger:GATed] {1 | 0}<NL>

HYSTeresis

Command :TRIGger:HYSTeresis {NORMal | HSENsitivity}

This command specifies the trigger hysteresis . NORMal is the typical hysteresis selection. HSENsitivity gives minimum hysteresis and the highest bandwidth.

Query :TRIGger:HYSTeresis?

The query returns the current hysteresis setting.

Returned Format [:TRIGger:HYSTeresis] {NORMal | HSENSitivity}<NL>

LEVel

Command :TRIGger:LEVel <level>

This command specifies the trigger level. Only one trigger level is stored in the analyzer.

<level> The trigger level on all trigger inputs.

Query :TRIGger:LEVel?

The query returns the trigger level.

Returned Format [:TRIGger:LEVel] <level> <NL>

SLOPe

Command :TRIGger:SLOPe {POSitive | NEGative}

This command specifies the slope of the edge on which to trigger.

Query :TRIGger:SLOPe?

The query returns the slope for the trigger.

Returned Format [:TRIGger:SLOPe] {POSitive | NEGative}<NL>

SOURce

Command :TRIGger:SOURce [<trigger> {FPANel | FRUN | LMODule | RMODule}]

This command selects the trigger input. Front Panel, Left Module, and Right Module are inputs from the front panel of the instrument. Free Run is internally generated, and is not affected by the settings of gates, level, slope, bandwidth, or hysteresis.

<trigger> Front PANel, Left MODule, and Right MODule are inputs on the front of the instrument. FreeRUN is internally generated and is unaffected by the settings for gated, level, slope, bandwidth or hysteresis.

Query :TRIGger:SOURce?

The query returns the current trigger source of the current mode.

Returned Format [:TRIGger:SOURce] <trigger><NL>

Waveform Commands

Waveform Commands

The WAVeform subsystem is used to transfer waveform data between a computer and the analyzer. It contains commands to set up the waveform transfer and to send or receive waveform records to or from the analyzer.

Data Acquisition

When the data is acquired using the DIGitize command, the data is placed in the channel or function memory of the specified source. After the DIGitize command, the analyzer is stopped. If the analyzer is restarted over GPIB or the front panel, the data acquired with the DIGitize command is overwritten.

You can query the preamble, elements of the preamble, or waveform data while the analyzer is running, but the data will reflect only the current acquisition, and subsequent queries will not reflect consistent data. For example, if the analyzer is running and you query the X origin, the data is queried in a separate GPIB command, and it is likely that the first point in the data will have a different time than that of the X origin. This is due to data acquisitions that may have occurred between the queries. For this reason, Agilent does not recommend this mode of operation. Instead, you should use the DIGitize command to stop the analyzer so that all subsequent queries will be consistent.

Function data is volatile and must be read following a DIGitize command or the data will be lost when the analyzer is turned off.

Waveform Data and Preamble

The waveform record consists of two parts: the preamble and the waveform data. The waveform data is the actual sampled data acquired for the specified source. The preamble contains the information for interpreting the waveform data, including the number of points acquired, the format of the acquired data, and the type of acquired data. The preamble also contains the X and Y increments, origins, and references for the acquired data.

The values in the preamble are set when you execute the DIGitize command. The preamble values are based on the settings of controls in the ACQuire subsystem.

Although you can change preamble values with a GPIB computer, you cannot change the way the data is acquired. Changing the preamble values cannot change the type of data that was actually acquired, the number of points actually acquired, etc.

CAUTION You must use extreme caution when changing any waveform preamble values to ensure that the data is still useful. For example, setting points in the preamble to a different value from the actual number of points in the waveform results in inaccurate data.

The waveform data and preamble must be read or sent using two separate commands: WAVeform:DATA and WAVeform:PREamble.

Data Conversion

Data sent from the analyzer must be scaled for useful interpretation. The values used to interpret the data are the X and Y origins, X and Y increments, and X and Y references. These values can be read from the waveform preamble.

Conversion from Data Value to Units

To convert the waveform data values (essentially A/D counts) to real-world units, such as volts, use the following scaling formulas:

Y-axis Units = (data value – Yreference) × Yincrement + Yorigin
X-axis Units = (data index – Xreference) × Xincrement + Xorigin,
where the data index starts at zero: 0, 1, 2, ..., n-1.

The first data point for the time (X-axis units) must be zero so the time of the first data point is the X origin.

Data Format for GPIB Transfer

There are four types of data formats that you can select with the WAVeform:FORMat command: ASCii, BYTE, WORD, and LONG. Refer to the FORMat command in this chapter for more information on data format.

Waveform Commands

BANDpass?

Query

:WAVeform:BANDpass?

This query returns an estimate of the maximum and minimum bandwidth limits of the source signal. Bandwidth limits are computed as a function of the coupling and the selected filter mode. Cutoff frequencies are derived from the acquisition path and software filtering.

Returned Format

[:WAVeform:BANDpass] <lower_cutoff>,<upper_cutoff><NL>

<lower_cutoff>	Minimum frequency passed by the acquisition system.
<upper_cutoff>	Maximum frequency passed by the acquisition system.

Example

This example places the estimated maximum and minimum bandwidth limits of the source signal in the string variable, Bandwidth$, then prints the contents of the variable to the computer's screen.

```
10 DIM Bandwidth$[50]                    !Dimension variable
20 OUTPUT 707;":WAVEFORM:BANDPASS?"
30 ENTER 707;Bandwidth$
40 PRINT Bandwidth$
50 END
```

BYTeorder

Command

:WAVeform:BYTeorder {MSBFirst | LSBFirst}

This command selects the order in which bytes are transferred to and from the analyzer using WORD and LONG formats. If MSBFirst is selected, the most significant byte is transferred first. Otherwise, the least significant byte is transferred first. The default setting is MSBFirst.

Example

This example sets up the analyzer to send the most significant byte first during data transmission.

```
10 OUTPUT 707;":WAVEFORM:BYTEORDER MSBFIRST"
20 END
```

Query

:WAVeform:BYTeorder?

The query returns the current setting for the byte order.

Returned Format

[:WAVeform:BYTeorder] {MSBFirst | LSBFirst}<NL>

Example

This example places the current setting for the byte order in the string variable, Setting$, then prints the contents of the variable to the computer screen.

```
10 DIM Setting$[10]                          !Dimension variable
20 OUTPUT 707;":WAVEFORM:BYTEORDER?"
30 ENTER 707;Setting$
40 PRINT Setting$
50 END
```

MSBFirst and LSBFirst

MSBFirst is for microprocessors, like Motorola's, where the most significant byte resides at the lower address. LSBFirst is for microprocessors, like Intel's, where the least significant byte resides at the lower address.

COUNt?

Query

:WAVeform:COUNt?

This query returns the fewest number of hits in all of the time buckets for the currently selected waveform. For the AVERAGE waveform type, the count value is the fewest number of hits for all time buckets. This value may be less than or equal to the value specified with the ACQuire:COUNt command.

For the NORMAL, RAW, INTERPOLATE, and VERSUS waveform types, the count value returned is one, unless the data contains holes (sample points where no data is acquired). If the data contains holes, zero is returned.

Returned Format

[:WAVeform:COUNt] <number><NL>

<number>

An integer. Values range from 1 to 262144 for NORMal, RAW, or INTerpolate types and from 1 to 32768 for VERSus type.

Example

This example places the current count field value in the string variable, Count$, then prints the contents of the variable to the computer's screen.

```
10 DIM Count$[50]                        !Dimension variable
20 OUTPUT 707;":WAVEFORM:COUNT?"
30 ENTER 707;Count$
40 PRINT Count$
50 END
```

DATA

Command

:WAVeform:DATA <block_data>[,<block_data>]

This command transfers waveform data to the analyzer over GPIB and stores the data in a previously specified waveform memory. The waveform memory is specified with the WAVeform:SOURce command. Only waveform memories and color grade/gray scale data may have waveform data sent to them. The format of the data being sent must match the format previously specified by the waveform preamble for the destination memory.

VERSus data is transferred as two arrays. The first array contains the data on the X axis, and the second array contains the data on the Y axis. The two arrays are transferred one at a time over GPIB in a linear format. There are *n* data points sent in each array, where *n* is the number in the points portion of the preamble.

The full-scale vertical range of the A/D converter will be returned with the data query. You should use the Y-increment, Y-origin, and Y-reference values to convert the full-scale vertical ranges to voltage values. You should use the Y-range and Y-display values to plot the voltage values. All of these reference values are available from the waveform preamble. Refer to "Conversion from Data Value to Units" earlier in this chapter.

<block_data>

Binary block data in the # format.

Example

This example sends 1000 bytes of previously saved data to the analyzer from the array, Set.

```
10 OUTPUT 707 USING "#,K";:WAVEFORM:DATA #800001000"
20 OUTPUT 707 USING "W";Set(*)
30 END
```

HP BASIC Image Specifiers

# is an HP BASIC image specifier that suppresses the automatic output of the EOL sequence following the last output item.

K is an HP BASIC image specifier that outputs a number or string in standard form with no leading or trailing blanks.

W is an HP BASIC image specifier that outputs 16-bit words with the most significant byte first.

Query

:WAVeform:DATA?

The query outputs waveform data to the computer over the GPIB interface. The data is copied from a waveform memory, function, or channel buffer previously specified with the WAVeform:SOURce command. The returned data is described by the waveform preamble.

Returned Format

[:WAVeform:DATA] <block_data>[,<block_data>]<NL>

Example

This example places the current waveform data from channel 1 of the array Wdata in the word format.

```
10 OUTPUT 707;":SYSTEM:HEADER OFF"              !Response headers off
20 OUTPUT 707;":WAVEFORM:SOURCE CHANNEL1        !Select source
30 OUTPUT 707;":WAVEFORM:FORMAT WORD"           !Select word format
40 OUTPUT 707;":WAVEFORM:DATA?"
50 ENTER 707 USING "#,1A";Pound_sign$
53 ENTER 707 USING "#,1D";Header_length
55 ENTER 707 USING "#,"&VAL$(Header_length)&"D";Length
60 Length = Length/2                            !Length in words
70 ALLOCATE INTEGER Wdata(1:Length)
80 ENTER 707 USING "#,W";Wdata(*)
90 ENTER 707 USING "-K,B";End$
100 END
```

HP BASIC Image Specifiers

# is an HP BASIC image specifier that terminates the statement when the last ENTER item is terminated. EOI and line feed are the item terminators.

1A is an HP BASIC image specifier that places the next character received in a string variable.

1D is an HP BASIC image specifier that places the next character in a numeric variable.

W is an HP BASIC image specifier that places the data in the array in word format with the first byte entered as the most significant byte.

-K is an HP BASIC image specifier that places the block data in a string, including carriage returns and line feeds until EOI is true or when the dimensioned length of the string is reached.

B is an HP BASIC specifier that enters the next byte in a variable.

The format of the waveform data must match the format previously specified by the WAVeform:FORMat, WAVeform:BYTeorder, and WAVeform:PREamble commands.

FORMat

Command

:WAVeform:FORMat {ASCii | BYTE | WORD}

This command sets the data transmission mode for waveform data output. This command controls how the data is formatted when the data is sent from the analyzer and pertains to all waveforms. The default format is ASCii.

ASCii

ASCII formatted data consists of ASCII digits with each data value separated by a comma. Data values can be converted to real values on the Y axis (for example, volts) and transmitted in floating point engineering notation. In ASCII:

- The value “99.999E+36” represents a hole level (a hole in the acquisition data).
- The value “99.999E+33” represents a clipped-high level.
- The value “99.999E+30” represents a clipped-low level.

BYTE

BYTE formatted data is formatted as signed 8-bit integers. If you use BASIC, you need to create a function to convert these signed bits to signed integers. In byte format:

- The value 125 represents a hole level (a hole in the acquisition data).
- The value 127 represents a clipped-high level.
- The value 126 represents a clipped-low level.

Data is rounded when converted from a larger size to a smaller size. For waveform transfer into the analyzer:

- The maximum valid qlevel is 124.
- The minimum valid qlevel is –128.

WORD

WORD formatted data is transferred as signed 16-bit integers in two bytes. If WAVeform:BYTeorder is set to MSBFirst, the most significant byte of each word is sent first. If the BYTeorder is LSBFirst, the least significant byte of each word is sent first. In word format:

- The value 31232 represents a hole level (no sample data at the current waveform data point).
- The value 32256 represents a clipped-high level.
- The value 31744 represents a clipped-low level.

For waveform transfer into the analyzer:

- The maximum valid qlevel is 30720.
- The minimum valid qlevel is –32736.

Example This example selects the WORD format for waveform data transmission.

```
10 OUTPUT 707;":WAVEFORM:FORMAT WORD"
20 END
```

Query :WAVeform:FORMat?

The query returns the current output format for transferring waveform data.

Returned Format [:WAVeform:FORMat] {ASCii | BYTE | LONG | WORD}<NL>

Example This example places the current output format for data transmission in the string variable, Mode$, then prints the contents of the variable to the computer screen.

```
10 DIM Mode$[50]                        !Dimension variable
20 OUTPUT 707;":WAVEFORM:FORMAT?"
30 ENTER 707;Mode$
40 PRINT Mode$
50 END
```

POINts?

Query :WAVeform:POINts?

The query returns the points value in the current waveform preamble. The points value is the number of time buckets contained in the waveform selected with the WAVeform:SOURce command.

Returned Format [:WAVeform:POINts] <points><NL>

<points> An integer. Values range from 1 to 262144. See the ACQuire:POINts command for more information.

Example This example places the current acquisition length in the numeric variable, Length, then prints the contents of the variable to the computer screen.

```
10 OUTPUT 707;":SYSTEM:HEADER OFF"      !Response headers off
20 OUTPUT 707;":WAVEFORM:POINTS?"
30 ENTER 707;Length
40 PRINT Length
50 END
```

Turn Headers Off

When you are receiving numeric data into numeric variables, you should turn the headers off. Otherwise, the headers may cause misinterpretation of returned data.

See Also The ACQuire:POINts command in the ACQuire Commands chapter.

PREamble

Command	:WAVeform:PREamble <preamble_data> This command sends a waveform preamble to the previously selected waveform memory in the analyzer. The preamble contains the scaling and other values used to describe the data. The waveform memory is specified with the WAVeform:SOURce command. Only waveform memories may have waveform data sent to them. The preamble can be used to translate raw data into time and voltage values. The following lists the elements in the preamble.
<preamble_data>	<format NR1>, <type NR1>, <points NR1>,<count NR1>, <X increment NR3>,<X origin NR3>, < X reference NR1>, <Y increment NR3>, <Y origin NR3>,<Y reference NR1>, <coupling NR1>, <X display range NR3>, <X display origin NR3>, <Y display range NR3>, <Y display origin NR3>, <date, string>, <time, string>, <frame model #, string>, <acquisition mode NR1>, <completion NR1>, <X units NR1>, <Y units NR1>, <max bandwidth limit NR3>, <min bandwidth limit NR3>
<date>	A string containing the data in the format DD MMM YYYY, where DD is the day, 1 to 31; MMM is the month; and YYYY is the year.
<time>	A string containing the time in the format HH:MM:SS:TT, where HH is the hour, 0 to 23, MM is the minute, 0 to 59, SS is the second, 0 to 59, and TT is the hundreds of seconds, 0 to 99.
<frame model #>	A string containing the model number and serial number of the frame in the format MODEL#:SERIAL#.
<format>	0 for ASCII format. 1 for BYTE format. 2 for WORD format.
<type>	1 for RAW type. 2 for AVERAGE type. 3 not used 4 not used 5 for VERSUS type. 6 not used 7 for NORMAL type. 8 for DATABASE type.
<acquisition mode>	2 for SEQUENTIAL mode.
<coupling>	0 for AC coupling.

<x units> **<y units>**	0 for UNKNOWN units. 1 for VOLT units. 2 for SECOND units. 3 for CONSTANT units. 4 for AMP units. 5 for DECIBEL units. 6 for HIT units. 7 for PERCENT units. 8 for WATT units.

See Table 26-1 on page 26-13 for descriptions of all the waveform preamble elements.

HP BASIC Image Specifiers

# is an HP BASIC image specifier that suppresses the automatic output of the EOL sequence following the last output item.

K is an HP BASIC image specifier that outputs a number or string in standard form with no leading or trailing blanks.

Query

:WAVeform:PREamble?

The query outputs a waveform preamble to the computer from the waveform source, which can be a waveform memory or channel buffer.

Returned Format

[:WAVeform:PREamble] <preamble_data><NL>

Example

This example outputs the current waveform preamble for the selected source to the string variable, Preamble$.

```
10 DIM Preamble$[250]                        !Dimension variable
20 OUTPUT 707;":SYSTEM:HEADER OFF"           !Response headers off
30 OUTPUT 707;":WAVEFORM:PREAMBLE?"
40 ENTER 707 USING "-K";Preamble$
50 END
```

Placing the Block in a String

-K is an HP BASIC image specifier that places the block data in a string, including carriage returns and line feeds, until EOI is true, or when the dimensioned length of the string is reached.

See Also

WAVeform:DATA

Table 26-1. Waveform Preamble Elements

Element	**Description**
Format	The format value describes the data transmission mode for waveform data output. This command controls how the data is formatted when it is sent from the analyzer. (See WAVeform:FORMat.)
Type	This value describes how the waveform was acquired. (See also WAVeform:TYPE.)
Points	The number of data points or data pairs contained in the waveform data. (See ACQuire:POINts.)
Count	For the AVERAGE waveform type, the count value is the minimum count or fewest number of hits for all time buckets. This value may be less than or equal to the value requested with the ACQuire:COUNt command. For NORMAL, RAW, INTERPOLATE, and VERSUS waveform types, this value is 0 or 1. The count value is ignored when it is sent to the analyzer in the preamble. (See WAVeform:TYPE and ACQuire:COUNt.)
X increment	The X increment is the duration between data points on the X axis. For time domain signals, this is the time between points. (See WAVeform:XINCrement.)
X Origin	The X origin is the X-axis value of the first data point in the data record. For time domain signals, it is the time of the first point. This value is treated as a double precision 64-bit floating point number. (See WAVeform:XORigin.)
X Reference	The X reference is the data point associated with the X origin. It is at this data point that the X origin is defined. In this analyzer, the value is always zero. (See WAVeform:XREFerence.)
Y Increment	The Y increment is the duration between Y-axis levels. For voltage waveforms, it is the voltage corresponding to one level. (See WAVeform:YINCrement.)
Y Origin	The Y origin is the Y-axis value at level zero. For voltage signals, it is the voltage at level zero. (See WAVeform:YORigin.)

Table 26-1. Waveform Preamble Elements (Continued)

Element	Description
Y Reference	The Y reference is the level associated with the Y origin. It is at this level that the Y origin is defined. In this analyzer, this value is always zero. (See WAVeform:YREFerence.)
Coupling	The input coupling of the waveform. The coupling value is ignored when sent to the analyzer in the preamble.
X Display Range	The X display range is the X-axis duration of the waveform that is displayed. For time domain signals, it is the duration of time across the display. (See WAVeform:XRANge.)
X Display Origin	The X display origin is the X-axis value at the left edge of the display. For time domain signals, it is the time at the start of the display. This value is treated as a double precision 64-bit floating point number. (See WAVeform:XDISplay.)
Y Display Range	The Y display range is the Y-axis duration of the waveform which is displayed. For voltage waveforms, it is the amount of voltage across the display. (See WAVeform:YRANge.)
Y Display Origin	(See WAVeform:YDISplay.)
Date	The date that the waveform was acquired or created.
Time	The time that the waveform was acquired or created.
Frame Model #	The model number of the frame that acquired or created this waveform. The frame model number is ignored when it is sent to an analyzer in the preamble.
Acquisition Mode	The acquisition sampling mode of the waveform.
Complete	The complete value is the percent of time buckets that are complete. The complete value is ignored when it is sent to the analyzer in the preamble. (See WAVeform:COMPlete.)
X Units	The X-axis units of the waveform. (See WAVeform:XUNits.)
Y Units	The Y-axis units of the waveform. (See WAVeform:YUNits.)

Table 26-1. Waveform Preamble Elements (Continued)

Element	Description
Band Pass	The band pass consists of two values that are an estimation of the maximum and minimum bandwidth limits of the source signal. The bandwidth limit is computed as a function of the selected coupling and filter mode. (See the WAVeform:BANDpass query.)

SOURce

Command

```
:WAVeform:SOURce {WMEMory<N> | FUNCtion<N> | CHANnel<N> | RESPonse<N> |
CGRade}
```

This command selects a channel, function, TDR response, waveform memory, or color grade/gray scale as the waveform source.

<N> For channels, waveform memories, TDR responses and functions: 1, 2, 3, or 4.

Example This example selects channel 1 as the waveform source.

```
10 OUTPUT 707;":WAVEFORM:SOURCE CHANNEL1"
20 END
```

Query

```
:WAVeform:SOURce?
```

The query returns the currently selected waveform source.

Returned Format

```
[:WAVeform:SOURce] {WMEMory<N> | FUNCtion<N> | RESPonse<N> | CHANnel<N> |
CGRade}<NL>
```

Example This example places the current selection for the waveform source in the string variable, Selection$, then prints the contents of the variable to the computer screen.

```
10 DIM Selection$[50]              !Dimension variable
20 OUTPUT 707;":WAVEFORM:SOURCE?"
30 ENTER 707;Selection$
40 PRINT Selection$
50 END
```

TYPE?

Query :WAVeform:TYPE?

This query returns the current acquisition data type for the currently selected source. The type returned describes how the waveform was acquired. The waveform type may be NORMAL, RAW, INTERPOLATE, AVERAGE, or VERSUS.

NORMAL Normal data consists of the last data point in each time bucket.

RAW Raw data consists of one data point in each time bucket with no interpolation.

INTERPOLATE In the interpolate acquisition type, the last data point in each time bucket is stored, and additional data points are filled in between the acquired data points by interpolation.

AVERAGE Average data consists of the average of the first *n* hits in a time bucket, where *n* is the value in the count portion of the preamble. Time buckets that have fewer than *n* hits return the average of the data they contain. 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.

VERSUS VERSus data consists of two arrays of data: one containing the X-axis values, and the other containing the Y-axis values. Versus waveforms can be generated using the FUNCtion subsystem commands.

Returned Format [:WAVeform:TYPE] {NORMal | RAW | INTerpolate | AVERage | VERSus}<NL>

Example This example places the current acquisition data type in the string variable, Type$, then prints the contents of the variable to the computer's screen.

```
10 DIM Type$[50]                    !Dimension variable
20 OUTPUT 707;":WAVEFORM:TYPE?"
30 ENTER 707;Type$
40 PRINT Type$
50 END
```

XDISplay?

Query :WAVeform:XDISplay?

This query returns the X-axis value at the left edge of the display. For time domain signals, it is the time at the start of the display. For VERSus type waveforms, it is the value at the center of the X-axis of the display. This value is treated as a double precision 64-bit floating point number.

Returned Format [:WAVeform:XDISplay] <value><NL>

<value> A real number representing the X-axis value at the left edge of the display.

Example This example returns the X-axis value at the left edge of the display to the numeric variable, Value, then prints the contents of the variable to the computer screen.

```
10 OUTPUT 707;":SYSTEM:HEADER OFF"!Response headers off
20 OUTPUT 707;":WAVEFORM:XDISPLAY?"
30 ENTER 707;Value
40 PRINT Value
50 END
```

XINCrement?

Query :WAVeform:XINCrement?

This query returns the duration between data points on the X axis. For time domain signals, this is the time difference between consecutive data points for the currently specified waveform source. For VERSus type waveforms, this is the duration between levels on the X axis. For voltage waveforms, this is the voltage corresponding to one level.

Returned Format [:WAVeform:XINCrement] <value><NL>

<value> A real number representing the duration between data points on the X axis.

Example This example places the current Xincrement value for the currently specified source in the numeric variable, Value, then prints the contents of the variable to the computer screen.

```
10 OUTPUT 707;":SYSTEM:HEADER OFF"           !Response headers off
20 OUTPUT 707;":WAVEFORM:XINCREMENT?"
30 ENTER 707;Value
40 PRINT Value
50 END
```

See Also You can obtain the Xincrement value through the WAVeform:PREamble query.

XORigin?

Query :WAVeform:XORigin?

This query returns the X-axis value of the first data point in the data record. For time domain signals, it is the time of the first point. For VERSus type waveforms, it is the X-axis value at level zero. For voltage waveforms, it is the voltage at level zero. The value returned by this query is treated as a double precision 64-bit floating point number.

Returned Format [:WAVeform:XORigin] <value><NL>

<value> A real number representing the X-axis value of the first data point in the data record.

Example This example places the current Xorigin value for the currently specified source in the numeric variable, Value, then prints the contents of the variable to the computer's screen.

```
10 OUTPUT 707;":SYSTEM:HEADER OFF"!Response headers off
20 OUTPUT 707;":WAVEFORM:XORIGIN?"
30 ENTER 707;Value
40 PRINT Value
50 END
```

See Also You can obtain the Xorigin value through the WAVeform:PREamble query.

XRANge?

Query :WAVeform:XRANge?

This query returns the X-axis duration of the displayed waveform. For time domain signals, it is the duration of the time across the display. For VERSus type waveforms, it is the duration of the waveform that is displayed on the X axis.

Returned Format [:WAVeform:XRANge] <value><NL>

<value> A real number representing the X-axis duration of the displayed waveform.

Example This example returns the X-axis duration of the displayed waveform to the numeric variable, Value, then prints the contents of the variable to the computer's screen.

```
10 OUTPUT 707;":SYSTEM:HEADER OFF"              !Response headers off
20 OUTPUT 707;":WAVEFORM:XRANGE?"
30 ENTER 707;Value
40 PRINT Value
50 END
```

XREFerence?

Query :WAVeform:XREFerence?

This query returns the data point or level associated with the Xorigin data value. It is at this data point or level that the X origin is defined. In this analyzer, the value is always zero.

Returned Format [:WAVeform:XREFerence] 0<NL>

Example This example places the current X reference value for the currently specified source in the numeric variable, Value, then prints the contents of the variable to the computer screen.

```
10 OUTPUT 707;":SYSTEM:HEADER OFF"          !Response headers off
20 OUTPUT 707;":WAVEFORM:XREFERENCE?"
30 ENTER 707;Value
40 PRINT Value
50 END
```

See Also You can obtain the Xreference value through the WAVeform:PREamble query.

XUNits?

Query :WAVeform:XUNits?

This query returns the X-axis units of the currently selected waveform source. The currently selected source may be a channel, function, or waveform memory.

Returned Format [:WAVeform:XUNits] {UNKNown | VOLT | SECond | CONStant | AMP | DECibels}<NL>

Example This example returns the X-axis units of the currently selected waveform source to the string variable, Unit$, then prints the contents of the variable to the computer's screen.

```
10 DIM Unit$[50]                            !Dimension variable
20 OUTPUT 707;":WAVEFORM:XUNITS?"
30 ENTER 707;Unit$
40 PRINT Unit$
50 END
```

YDISplay?

Query :WAVeform:YDISplay?

This query returns the Y-axis value at the center of the display. For voltage signals, it is the voltage at the center of the display.

Returned Format [:WAVeform:YDISplay] <value><NL>

<value> A real number representing the Y-axis value at the center of the display.

Example

This example returns the current Y display value to the numeric variable, Value, then prints the contents of the variable to the computer screen.

```
10 OUTPUT 707;":SYSTEM:HEADER OFF"      !Response headers off
20 OUTPUT 707;":WAVEFORM:YDISPLAY?"
30 ENTER 707;Value
40 PRINT Value
50 END
```

YINCrement?

Query

:WAVeform:YINCrement?

This query returns the duration between the Y-axis levels.

- For BYTE and WORD data, and voltage waveforms, it is the voltage corresponding to one level.
- For ASCII data format, the YINCrement is the full voltage range covered by the A/D converter.

Returned Format

[:WAVeform:YINCrement] <real_value><NL>

<real_value>

A real number in exponential (NR3) format.

Example

This example places the current Yincrement value for the currently specified source in the numeric variable, Value, then prints the contents of the variable to the computer's screen.

```
10 OUTPUT 707;":SYSTEM:HEADER OFF"            !Response headers off
20 OUTPUT 707;":WAVEFORM:YINCREMENT?"
30 ENTER 707;Value
40 PRINT Value
50 END
```

See Also

You can obtain the Yincrement value through the WAVeform:PREamble query.

YORigin?

Query

:WAVeform:YORigin?

This query returns the Y-axis value at level zero.

- For BYTE and WORD data, and voltage signals, it is the voltage at level zero.
- For ASCII data format, the YORigin is the Y-axis value at the center of the data range. Data range is returned in the Y increment.

Returned Format

[:WAVeform:YORigin] <real_value><NL>

<real_value> A real number in exponential (NR3) format.

Example This example places the current Y origin value in the numeric variable, Center, then prints the contents of the variable to the computer screen.

```
10 OUTPUT 707;":SYSTEM:HEADER OFF"      !Response headers off
20 OUTPUT 707;":WAVEFORM:YORIGIN?"
30 ENTER 707;Center
40 PRINT Center
50 END
```

See Also You can obtain the YORigin value through the WAVeform:PREamble query.

YRANge?

Query :WAVeform:YRANge?

This query returns the Y-axis duration of the displayed waveform. For voltage signals, it is the voltage across the entire display.

Returned Format [:WAVeform:YRANge] <value><NL>

<value> A real number representing the Y-axis duration of the displayed waveform.

Example This example returns the current Y Range value to the numeric variable, Value, then prints the contents of the variable to the computer's screen.

```
10 OUTPUT 707;":SYSTEM:HEADER OFF"      !Response headers off
20 OUTPUT 707;":WAVEFORM:YRANGE?"
30 ENTER 707;Value
40 PRINT Value
50 END
```

YREFerence?

Query :WAVeform:YREFerence?

This query returns the level associated with the Y origin. It is at this level that the Y origin is defined. In this analyzer, the value is always zero.

Returned Format [:WAVeform:YREFerence] <integer_value><NL>

<integer_value> Always 0.

Example

This example places the current Y Reference value for the currently specified source in the numeric variable, Value, then prints the contents of the variable to the computer's screen.

```
10 OUTPUT 707;":SYSTEM:HEADER OFF"      !Response headers off
20 OUTPUT 707;":WAVEFORM:YREFERENCE?"
30 ENTER 707;Value
40 PRINT Value
50 END
```

See Also

You can obtain the YReference value through the WAVeform:PREamble query.

YUNits?

Query

:WAVeform:YUNits?

This query returns the Y-axis units of the currently selected waveform source. The currently selected source may be a channel, function, or waveform memory.

Returned Format

[:WAVeform:YUNits] {UNKNown | VOLT | SECond | CONStant | AMP | WATT}<NL>

Example

This example returns the Y-axis units of the currently selected waveform source to the string variable, Unit$, then prints the contents of the variable to the computer's screen.

```
10 DIM Unit$[50]                    !Dimension variable
20 OUTPUT 707;":WAVEFORM:YUNITS?"
30 ENTER 707;Unit$
40 PRINT Unit$
50 END
```

27

Waveform Memory Commands

Waveform Memory Commands

The Waveform Memory Subsystem commands allow you to save and display waveforms, memories, and functions.

<N> Indicates the Waveform Memory Number

In Waveform Memory commands, the <N> in WMEMory<N> represents the waveform memory number (1-4).

DISPlay

Command :WMEMory<N>:DISPlay {{ON|1}|{OFF|0}}

This command enables or disables the viewing of the selected waveform memory.

<N> The memory number is an integer from 1 to 4.

Example This example turns on the waveform memory 1 display.

```
10 OUTPUT 707;":WMEMORY1:DISPLAY ON"
20 END
```

Query :WMEMory<N>:DISPlay?

The query returns the state of the selected waveform memory.

Returned Format [:WMEMory<N>:DISPlay] {1 | 0}<NL>

LOAD

Command :WMEMory<N>:LOAD <file_name>

This command loads an analyzer waveform memory location with a waveform from a file which has an internal waveform format (extension .wfm) or a verbose/yvalues waveform format (extension .txt). You can load the file either from the C:\ drive or A:\ drive. See the examples below.

The scope assumes the default path for waveforms is C:\User Files\Waveforms. To use a different path, please specify the path and file name completely.

<N> The memory number is an integer from 1 to 4.

<file_name> Specifies the file to load, and has either a .wfm or .txt extension.

Examples This example loads waveform memory 4 with a file that has the internal waveform format.

```
10 OUTPUT 707;":WMEMORY4:LOAD ""c:\User Files\Waveforms\waveform.wfm"""
20 END
```

This example loads waveform memory 3 with a file on the floppy drive that has the internal waveform format.

```
10 OUTPUT 707;":WMEMORY3:LOAD ""a:\waveform.wfm"""
20 END
```

Related Commands DISK:LOAD, DISK:STORe

SAVE

Command :WMEMory<N>:SAVE {CHANnel<N> | WMEMory<N> | FUNCtion<N> | RESPonse<N>}

This command stores the specified channel, waveform memory, TDR response, or function to the waveform memory. The channel or function must be displayed (DISPlay set to ON) or an error status is returned. You can save waveforms to waveform memories whether the waveform memory is displayed or not.

<N> An integer from 1 to 4.

Example This example saves channel 1 to waveform memory 4.

```
10 OUTPUT 707;":WMEMORY4:SAVE chan1"
20 END
```

XOFFset

Command :WMEMory<N>:XOFFset <offset_value>

This command sets the x-axis, horizontal position for the selected waveform memory's display scale. Position is referenced to center screen.

<N> The memory number is an integer from 1 to 4.

<offset_value> The horizontal offset (position) value.

Example This example sets the x-axis, horizontal position for waveform memory 3 to 0.1 seconds (100 ms).

```
10 OUTPUT 707;":WMEMORY3:XOFFSET 0.1"
20 END
```

Query :WMEMory<N>:XOFFset?

The query returns the current x-axis, horizontal position for the selected waveform memory.

Returned Format [:WMEMory<N>:XOFFset] <offset_value><NL>

XRANge

Command :WMEMory<N>:XRANge <range_value>

This command sets the x-axis, horizontal range for the selected waveform memory's display scale. The horizontal scale is the horizontal range divided by 10.

<N>	The memory number is an integer from 1 to 4.
<range_value>	The horizontal range value.
Example	This example sets the x-axis, horizontal range of waveform memory 2 to 435 microseconds.

```
10 OUTPUT 707;":WMEMORY2:XRANGE 435E-6"
20 END
```

Query :WMEMory<N>:XRANge?

The query returns the current x-axis, horizontal range for the selected waveform memory.

Returned Format [:WMEMory<N>:XRANge] <range_value><NL>

YOFFset

Command :WMEMory<N>:YOFFset <offset_value>

This command sets the y-axis (vertical axis) offset for the selected waveform memory.

<N>	The memory number is an integer from 1 to 4.
<offset_value>	The vertical offset value.
Example	This example sets the y-axis (vertical) offset of waveform memory 2 to 0.2V.

```
10 OUTPUT 707;":WMEMORY2:YOFFSET 0.2"
20 END
```

Query :WMEMory<N>:YOFFset?

The query returns the current y-axis (vertical) offset for the selected waveform memory.

Returned Format [:WMEMory<N>:YOFFset] <offset_value><NL>

YRANge

Command :WMEMory<N>:YRANge <range_value>

This command sets the y-axis, vertical range for the selected memory. The vertical scale is the vertical range divided by 8.

<N> The memory number is an integer from 1 to 4.

<range_value> The vertical range value.

Example This example sets the y-axis (vertical) range of waveform memory 3 to 0.2 volts.

```
10 OUTPUT 707;":WMEMORY3:YRANGE 0.2"
20 END
```

Query :WMEMory<N>:YRANge?

The query returns the Y-axis, vertical range for the selected memory.

Returned Format [:WMEMory<N>:YRANge] <range_value><NL>

Language Compatibility

Agilent 83480A Commands Not Used in the Agilent 86100A

Agilent 83480A/54750A Programming Commands Not Used in the 86100A (1 of 3)

Calibration Commands :CALibrate	
:FRAMe:CANCel	:PLUGin:MEMory?
:FRAMe:CONTinue	:PLUGin:OFFSet
:FRAMe:DATA	:PLUGin:OPOWer
:FRAMe:DONE?	:PLUGin:OPTical
:FRAMe:MEMory?	:PLUGin:OWAVelength
:PLUGin:ACCuracy	:PLUGin:TIME?
:PLUGin:CANCel	:PLUGin:VERTical
:PLUGin:CONTinue	:PROBe
:PLUGin:DONE?	:SAMPlers (added with plug-in)
Channel Commands :CHANnel	
:SKEW	
:UNITs:ATTenuation	
:UNITs:OFFSet	
Disk Commands :DISK	
:FORMat	
Display Commands :DISPlay	
:ASSign (added with plug-in)	:LABel
:CGRade	:MASK
:DWAVeform	:SOURce
:FORMat	:TEXT BLANk
:INVerse	

Agilent 83480A/54750A Programming Commands Not Used in the 86100A (2 of 3)

FFT Commands :FFT	
:DISPlay	:RANGe
:OFFSet	:SOURce
Hardcopy Commands :HARDcopy	
:ADDRess	:FFEed
:BACKground	:FILename
:DESTination	:LENGth
:DEVice	:MEDia
Limit Test Commands :LTESt	
:SSCReen:DDISk	:SSCReen:DPRinter:PFORmat
:SSCReen:DDISk:BACKground	:SSCReen:DPRinter:PORT
:SSCReen:DDISk:MEDia	:SSUMmary:ADDRess
:SSCReen:DDISk:PFORmat	:SSUMmary:FORmat
:SSCReen:DPRinter	:SSUMmary:MEDia
:SSCReen:DPRinter:ADDRess	:SSUMmary:PFORmat
:SSCReen:DPRinter:BACKground	:SSUMmary:PORT
:SSCReen:DPRinter:MEDia	
Mask Test Commands :MTESt	
:AMASk:CReate	:SSCReen:DDISk:MEDia
:AMASk:SOURce	:SSCReen:DDISk:PFORmat
:AMASk:UNITs	:SSCReen:DPRinter
:AMASk:XDELta	:SSCReen:DPRinter:ADDRess
:AMASk:YDELta	:SSCReen:DPRinter:BACKground
:AMODe	:SSCReen:DPRinter:MEDia
:DEFine	:SSCReen:DPRinter:PFORmat
:FENable	:SSCReen:DPRinter:PORT
:POLYgon:DEFine	:SSUMmary:ADDRess
:RECall	:SSUMmary:MEDia
:SAVE	:SSUMmary:PFORmat
:SSCReen:DDISk	:SSUMmary:PORT
:SSCReen:DDISk:BACKground	:SSUMmary:STANdard

Agilent 83480A/54750A Programming Commands Not Used in the 86100A (3 of 3)

Measure Commands :MEASure	
:FFT	
Pixel Memory Commands :PMEMory1	
:ADD	:MERGe
:CLEar	:RECall
:DISPlay	:STORe
:ERASe	
Service Commands :SERVice	
:COMMents	:DECLassify
System Commands :SYSTem	
:KEY	
Timebase Commands :TIMebase	
:VIEW	:WINDow:RANGe
:WINDow:DELay	:WINDow:SCALe
:WINDow:POSition	:WINDow:SOURce

Error Messages

Error Messages

This chapter describes the error messages and how they are generated. The possible causes for the generation of the error messages are also listed in Table 29-1 on page 29-6.

Error Queue

As errors are detected, they are placed in an error queue. This queue is first in, first out. If the error queue overflows, the last error in the queue is replaced with error –350, “Queue overflow.” Anytime the error queue overflows, the oldest errors remain in the queue, and the most recent error is discarded. The length of the analyzer's error queue is 30 (29 positions for the error messages, and 1 position for the “Queue overflow” message). Reading an error from the head of the queue removes that error from the queue, and opens a position at the tail of the queue for a new error. When all errors have been read from the queue, subsequent error queries return 0, “No error.”

The error queue is cleared when any of the following occur:

- the instrument is powered up,
- a *CLS command is sent,
- the last item from the queue is read, or
- the instrument is switched from talk only to addressed mode on the front panel.

Error Numbers

The error numbers are grouped according to the type of error that is detected.

- +0 indicates no errors were detected.
- –100 to –199 indicates a command error was detected.
- –200 to –299 indicates an execution error was detected.
- –300 to –399 indicates a device-specific error was detected.
- –400 to –499 indicates a query error was detected.
- +1 to +32767 indicates an analyzer-specific error has been detected.

Command Error

An error number in the range –100 to –199 indicates that an IEEE 488.2 syntax error has been detected by the instrument's parser. The occurrence of any error in this class sets the command error bit (bit 5) in the event status register and indicates that one of the following events occurred:

- An IEEE 488.2 syntax error was detected by the parser. That is, a controller-to-analyzer message was received that is in violation of the IEEE 488.2 standard. This may be a data element that violates the analyzer's listening formats, or a data type that is unacceptable to the analyzer.
- An unrecognized header was received. Unrecognized headers include incorrect analyzer-specific headers and incorrect or unimplemented IEEE 488.2 common commands.
- A Group Execute Trigger (GET) was entered into the input buffer inside of an IEEE 488.2 program message.

Events that generate command errors do not generate execution errors, analyzer-specific errors, or query errors.

Execution Error

An error number in the range –200 to –299 indicates that an error was detected by the instrument's execution control block. The occurrence of any error in this class causes the execution error bit (bit 4) in the event status register to be set. It also indicates that one of the following events occurred:

- The program data following a header is outside the legal input range or is inconsistent with the analyzer's capabilities.
- A valid program message could not be properly executed due to some analyzer condition.

Execution errors are reported by the analyzer after expressions are evaluated and rounding operations are completed. For example, rounding a numeric data element will not be reported as an execution error. Events that generate execution errors do not generate command errors, analyzer specific errors, or query errors.

Device- or Analyzer-Specific Error

An error number in the range of –300 to –399 or +1 to +32767 indicates that the instrument has detected an error caused by an analyzer operation that did not properly complete. This may be due to an abnormal hardware or firmware condition. For example, this error may be generated by a self-test response error, or a full error queue. The occurrence of any error in this class causes the analyzer-specific error bit (bit 3) in the event status register to be set.

Query Error

An error number in the range –400 to –499 indicates that the output queue control of the instrument has detected a problem with the message exchange protocol. An occurrence of any error in this class causes the query error bit (bit 2) in the event status register to be set. An occurrence of an error also means one of the following is true:

- An attempt is being made to read data from the output queue when no output is either present or pending.
- Data in the output queue has been lost.

List of Error Messages

Table 29-1 is a list of the error messages that are returned by the parser on this analyzer.

Table 29-1. Error Messages

0	No error	The error queue is empty. Every error in the queue has been read (SYSTEM:ERROR? query) or the queue was cleared by power-up or *CLS.
-100	Command error	This is the generic syntax error used if the analyzer cannot detect more specific errors.
-101	Invalid character	A syntactic element contains a character that is invalid for that type.
-102	Syntax error	An unrecognized command or data type was encountered.
-103	Invalid separator	The parser was expecting a separator and encountered an illegal character.
-104	Data type error	The parser recognized a data element different than one allowed. For example, numeric or string data was expected but block data was received.
-105	GET not allowed	A Group Execute Trigger was received within a program message.
-108	Parameter not allowed	More parameters were received than expected for the header.
-109	Missing parameter	Fewer parameters were received than required for the header.
-112	Program mnemonic too long	The header or character data element contains more than twelve characters.
-113	Undefined header	The header is syntactically correct, but it is undefined for the analyzer. For example, *XYZ is not defined for the analyzer.
-121	Invalid character in number	An invalid character for the data type being parsed was encountered. For example, a “9” in octal data.
-123	Numeric overflow	Number is too large or too small to be represented internally.
-124	Too many digits	The mantissa of a decimal numeric data element contained more than 255 digits excluding leading zeros.
-128	Numeric data not allowed	A legal numeric data element was received, but the analyzer does not accept one in this position for the header.
-131	Invalid suffix	The suffix does not follow the syntax described in IEEE 488.2 or the suffix is inappropriate for the analyzer.
-138	Suffix not allowed	A suffix was encountered after a numeric element that does not allow suffixes.
-141	Invalid character data	Either the character data element contains an invalid character or the particular element received is not valid for the header.
-144	Character data too long	
-148	Character data not allowed	A legal character data element was encountered where prohibited by the analyzer.
-150	String data error	This error can be generated when parsing a string data element. This particular error message is used if the analyzer cannot detect a more specific error.

Table 29-1. Error Messages (Continued)

-151	Invalid string data	A string data element was expected, but was invalid for some reason. For example, an END message was received before the terminal quote character.
-158	String data not allowed	A string data element was encountered but was not allowed by the analyzer at this point in parsing.
-160	Block data error	This error can be generated when parsing a block data element. This particular error message is used if the analyzer cannot detect a more specific error.
-161	Invalid block data	
-168	Block data not allowed	A legal block data element was encountered but was not allowed by the analyzer at this point in parsing.
-170	Expression error	This error can be generated when parsing an expression data element. It is used if the analyzer cannot detect a more specific error.
-171	Invalid expression	
-178	Expression data not allowed	Expression data was encountered but was not allowed by the analyzer at this point in parsing.
-200	Execution error	This is a generic syntax error which is used if the analyzer cannot detect more specific errors.
-220	Parameter error	Indicates that a program data element related error occurred.
-221	Settings conflict	Indicates that a legal program data element was parsed but could not be executed due to the current device state.
-222	Data out of range	Indicates that a legal program data element was parsed but could not be executed because the interpreted value is outside the legal range defined by the analyzer.
-223	Too much data	Indicates that a legal program data element of block, expression, or string type was received that contained more data than the analyzer could handle due to memory or related analyzer-specific requirements.
-224	Illegal parameter value	
-225	Out of memory	The device has insufficient memory to perform the requested operation.
-231	Data questionable	Indicates that measurement accuracy is suspect.
-240	Hardware error	Indicates that a legal program command or query could not be executed because of a hardware problem in the device.
-241	Hardware missing	
-250	Mass storage error	Indicates that a mass storage error occurred.
-251	Missing mass storage	Indicates that a legal program command or query could not be executed because of missing mass storage; for example, an option that was not installed.
-252	Missing media	Indicates that a legal program command or query could not be executed because of a missing media; for example, no disk.
-253	Corrupt media	Indicates that a legal program command or query could not be executed because of corrupt media; for example, bad disk or wrong format.

Table 29-1. Error Messages (Continued)

-254	Media full	Indicates that a legal program command or query could not be executed because the media was full; for example, there is no room on the disk.
-255	Directory full	Indicates that a legal program command or query could not be executed because the media directory was full.
-256	File name not found	Indicates that a legal program command or query could not be executed because the file name on the device media was not found; for example, an attempt was made to read or copy a nonexistent file.
-257	File name error	Indicates that a legal program command or query could not be executed because the file name on the device media was in error; for example, an attempt was made to copy to a duplicate file name.
-258	Media protected	Indicates that a legal program command or query could not be executed because the media was protected; for example, the write-protect tab on a disk was present.
-300	Service specific error	
-310	System error	Indicates that a system error occurred.
-350	Queue overflow	Indicates that there is no room in the error queue and an error occurred but was not recorded.
-400	Query error	This is the generic query error.
-410	Query INTERRUPTED	
-420	Query UNTERMINATED	
-430	Query DEADLOCKED	
-440	Query UNTERMINATED after indefinite response	

Index