HEWLETT PACKARD

Programmer's Guide

HP 1660-Series 100-MHz State/500-MHz Timing Logic Analyzers

Programmer's Guide

Publication number 01660-90902
First edition, August 1992

HP 1660-Series Logic Analyzers

In This Guide

This HP 1660 series programmer's guide is organized in three parts to make accessing information easy and logical. Part 1 consists of chapters 1 through 7 and contains general information about programming basics, HP-IB and RS-232C interface requirements, documentation conventions, status reporting, and error messages. If you are already familiar with IEEE 488.2 programming and HP-IB or RS-232C, you may want to just scan these chapters. If you are new to programming logic analyzers you should read part 1.

Chapter 1 is divided into two sections. The first section, "Talking to the Instrument," concentrates on program syntax, and the second section, "Receiving Information from the Instrument," discusses how queries are sent and how to retrieve query results from the instrument. Read either chapter 2, "Programming Over HP-IB," or chapter 3, "Programming Over RS-232C" for information concerning the physical connection between the HP 1660A-series logic analyzers and your controller. Chapter 4, "Programming and Documentation Conventions," gives an overview of all instructions and also explains the notation conventions used in the syntax definitions and examples. Chapter 5, "Message Communication and System Functions," provides an overview of the operation of instruments that operate in compliance with the IEEE 488.2 standard. Chapter 6 explains status reporting and how it can be used to monitor the flow of your programs and measurement process. Chapter 7 contains error message descriptions.

Part 2, chapters 8 through 26, explain each command in the command set for the logic analyzer. These chapters are organized in subsystems with each subsystem representing a front-panel menu.

The instructions listed in this part give you access to the measurements and front-panel features of the logic analyzers. The complexity of your programs and the tasks they accomplish are limited only by your imagination. This part is designed to provide a concise description of each instruction.

Part 3, chapter 27, contains program examples of actual tasks that show you how to get started in programming the HP 1660-series logic analyzers. These examples are written in HP Basic 6.2; however, the program concepts can be used in any other popular programming language that allows communications with either the HP-IB or RS-232C buses.

Contents

11 MMEMory Subsystem

12 INTermodule Subsystem

13 MACHine Subsystem

14 WLISt Subsystem

15 SFORmat Subsystem

19 SCHart Subsystem

20 COMPare Subsystem

21 TFORmat Subsystem

22 TTRigger (TTRace) Subsystem

23 TWAVeform Subsystem

1

Introduction to Programming

Introduction

This chapter introduces you to the basics of remote programming and is organized in two sections. The first section, "Talking to the Logic analyzer," concentrates on initializing the bus, program syntax and the elements of a syntax instuction. The second section, "Receiving Information from the Logic analyzer," discusses how queries are sent and how to retrieve query results from the logic analyzer.

The programming instructions explained in this book conform to IEEE Std 488.2-1987, "IEEE Standard Codes, Formats, Protocols, and Common Commands." These programming instructions provide a means of remotely controlling the HP 1660-series logic analyzers. There are three general categories of use. You can:

- Set up the logic analyzer and start measurements
- Retrieve setup information and measurement results
- Send measurement data to the logic analyzer

The instructions listed in this manual give you access to the measurements and front panel features of the HP 1660-series. The complexity of your programs and the tasks they accomplish are limited only by your imagination. This programming reference is designed to provide a concise description of each instruction.

Talking to the Instrument

In general, computers acting as controllers communicate with the instrument by sending and receiving messages over a remote interface, such as HP-IB or RS-232C. Instructions for programming the HP 1660-series will normally appear as ASCII character strings embedded inside the output statements of a "host" language available on your controller. The host language's input statements are used to read in responses from the HP 1660-series.

For example, HP 9000 Series 200/300 Basic uses the OUTPUT statement for sending commands and queries to the HP 1660-series. After a query is sent, the response can be read in using the ENTER statement. All programming examples in this manual are presented in HP Basic.

Example

This Basic statement sends a command that causes the logic analyzer's machine 1 to be a state analyzer:

```
OUTPUT XXX;":MACHINE1:TYPE STATE" <terminator>
```

Each part of the above statement is explained in this section.

Initialization

To make sure the bus and all appropriate interfaces are in a known state, begin every program with an initialization statement. Basic provides a CLEAR command that clears the interface buffer. If you are using HP-IB, CLEAR will also reset the parser in the logic analyzer. The parser is the program resident in the logic analyzer that reads the instructions you send to it from the controller.

After clearing the interface, you could preset the logic analyzer to a known state by loading a predefined configuration file from the disk.

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

Example

This Basic statement would load the configuration file "DEFAULT " (if it exists) into the logic analyzer.

```
OUTPUT XXX;":MMEMORY:LOAD:CONFIG 'DEFAULT  '"
```

Refer to chapter 10, "MMEMory Subsystem" for more information on the LOAD command.

Example Program

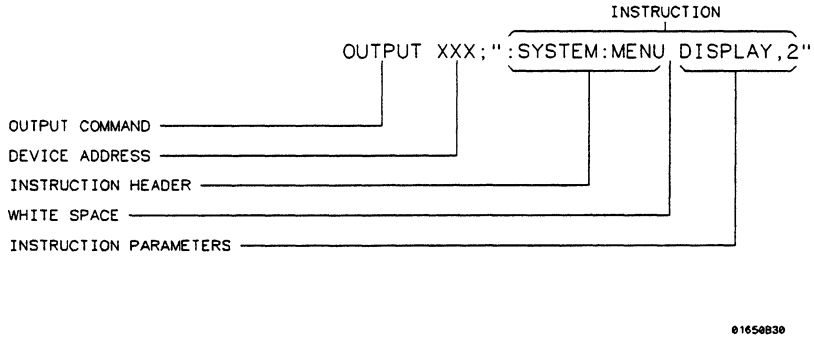
This program demonstrates the basic command structure used to program the HP 1660-series logic analyzers.

```
10 CLEAR XXX                                !Initialize instrument interface
20 OUTPUT XXX;":SYSTEM:HEADER ON"           !Turn headers on
30 OUTPUT XXX;":SYSTEM:LONGFORM ON"         !Turn longform on
40 OUTPUT XXX;":MMEM:LOAD:CONFIG 'TEST E'"  !Load configuration file
50 OUTPUT XXX;":MENU FORMAT,1"              !Select Format menu for machine 1
60 OUTPUT XXX;":RMODE SINGLE"               !Select run mode
70 OUTPUT XXX;":START"                      !Run the measurement
```

Instruction Syntax

To program the logic analyzer remotely, you must have an understanding of the command format and structure. The IEEE 488.2 standard governs syntax rules pertaining to how individual elements, such as headers, separators, parameters and terminators, may be grouped together to form complete instructions. Syntax definitions are also given to show how query responses will be formatted. Figure 1-1 shows the three main syntactical parts of a typical program statement: Output Command, Device Address, and Instruction. The instruction is further broken down into three parts: Instruction header, White space, and Instruction parameters.

Figure 1-1



Program Message Syntax

Output Command

The output command depends on the language you choose to use. Throughout this guide, HP 9000 Series 200/300 Basic 6.2 is used in the programming examples. If you use another language, you will need to find the equivalents of Basic Commands, like OUTPUT, ENTER and CLEAR in order to convert the examples. The instructions are always shown between the double quotes.

Device Address

The location where the device address must be specified also depends on the host language that you are using. In some languages, this could be specified outside the output command. In Basic, this is always specified after the keyword OUTPUT. The examples in this manual use a generic address of XXX. When writing programs, the number you use will depend on the cable you use, in addition to the actual address. If you are using an HP-IB, see chapter 2, "Programming over HP-IB." If you are using RS-232C, see chapter 3, "Programming Over RS-232C."

Instructions

Instructions (both commands and queries) normally appear as a string embedded in a statement of your host language, such as 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>. There are just a few instructions which use block data.

Instructions are composed of two main parts: the header, which specifies the command or query to be sent; and the parameters, which provide additional data needed to clarify the meaning of the instruction. Many queries do not use any parameters.

Instruction Header

The instruction header is one or more keywords separated by colons (:). The command tree in figure 4-1 illustrates how all the keywords can be joined together to form a complete header (see chapter 4, "Programming and Documentation Conventions").

The example in figure 1-1 shows a command. Queries are indicated 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 have included the question mark. The command and query forms of an instruction usually have different parameters.

When you look up a query in this programmer's reference, you'll find a paragraph labeled "Returned Format" under the one labeled "Query." The syntax definition by "Returned format" will always show the instruction header in square brackets, like [:SYSTem:MENU], which means the text between the brackets is optional. It is also a quick way to see what the header looks like.

White Space

White space is used to separate the instruction header from the instruction parameters. If the instruction does not use any parameters, white space does not need to be included. White space is defined as one or more spaces. ASCII defines a space to be a character, represented by a byte, that has a decimal value of 32. Tabs can be used only if your controller first converts them to space characters before sending the string to the logic analyzer.

Instruction Parameters

Instruction parameters are used to clarify the meaning of the command or query. They provide necessary data, such as: whether a function should be on or off, which waveform is to be displayed, or which pattern is to be looked for. Each instruction's syntax definition shows the parameters, as well as the range of acceptable values they accept. This chapter's "Parameter Data Types" section has all of the general rules about acceptable values.

When there is more than one parameter, they are separated by commas (,). White space surrounding the commas is optional.

Instruction Terminator

An instruction is executed after the instruction terminator is received. The terminator is the NL (New Line) character. The NL character is an ASCII linefeed character (decimal 10).

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

Header Types

There are three types of headers: Simple Command, Compound Command, and Common Command.

Simple Command Header

Simple command headers contain a single keyword. START and STOP are examples of simple command headers typically used in this logic analyzer. The syntax is: `<function><terminator>`

When parameters (indicated by <data>) must be included with the simple command header, the syntax is: `<function><white_space><data> <terminator>`

Example

```
:RMODE SINGLE<terminator>
```

Compound Command Header

Compound command headers are a combination of two or more program keywords. The first keyword selects the subsystem, and the last keyword selects the function within that subsystem. Sometimes you may need to list more than one subsystem before being allowed to specify the function. The keywords within the compound header are separated by colons. For example, to execute a single function within a subsystem, use the following:
`:<subsystem>:<function><white_space><data><terminator>`

Example

```
:SYSTEM:LONGFORM ON
```

To traverse down one level of a subsystem to execute a subsystem within that subsystem, use the following:
`<subsystem>:<subsystem>:<function><white_space> <data><terminator>`

Example

```
:MMEMORY:LOAD:CONFIG "FILE  "
```

Common Command Header

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

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

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

Combined Commands in the Same Subsystem

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

```
:<subsystem>:<function><white
space><data>;<function><white space><data><terminator>
```

Example

```
:SYSTEM:LONGFORM ON;HEADER ON
```

Duplicate Keywords

Identical function keywords can be used for more than one subsystem. For example, the function keyword MMODE may be used to specify the marker mode in the subsystem for state listing or the timing waveforms:

- `:SLIST:MMODE PATTERN` - sets the marker mode to pattern in the state listing.
- `:TWAVEFORM:MMODE TIME` - sets the marker mode to time in the timing waveforms.

SLIST and TWAVEFORM are subsystem selectors, and they determine which marker mode is being modified.

Query Usage

Logic analyzer instructions that are immediately followed by a question mark (?) are queries. After receiving a query, the logic analyzer parser places the response in the output buffer. The output message remains in the buffer until it is read or until another logic analyzer instruction is issued. When read, the message is transmitted across the bus to the designated listener (typically a controller).

Query commands are used to find out how the logic analyzer is currently configured. They are also used to get results of measurements made by the logic analyzer.

Example

This instruction places the current full-screen time for machine 1 in the output buffer.

```
:MACHINE1:TWAVEFORM:RANGE?
```

In order to prevent the loss of data in the output buffer, the output buffer must be read before the next program message is sent. Sending another command before reading the result of the query will cause the output buffer to be cleared and the current response to be lost. This will also generate a "QUERY UNTERMINATED" error in the error queue. For example, when you send the query `:TWAVEFORM:RANGE?` you must follow that with an input statement. In Basic, this is usually done with an ENTER statement.

In Basic, the input statement, `ENTER XXX; Range`, passes the value across the bus to the controller and places it in the variable Range.

Additional details on how to use queries is in the next section of this chapter, "Receiving Information for the Logic analyzer."

Program Header Options

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

Both program command and query headers may be sent in either long form (complete spelling), short form (abbreviated spelling), or any combination of long form and short form.

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

The rules for short form syntax are discussed in chapter 4, "Programming and Documentation Conventions."

Example

Either of the following examples turns on the headers and long form.
Long form:

```
OUTPUT XXX;":SYSTEM:HEADER ON;LONGFORM ON"
```

Short form:

```
OUTPUT XXX;":SYST:HEAD ON;LONG ON"
```

Parameter Data Types

There are three main types of data which are used in parameters. They are numeric, string, and keyword. A fourth type, block data, is used only for a few instructions: the DATA and SETup instructions in the SYSTem subsystem (see chapter 6); the CATalog, UPLoad, and DOWNload instructions in the MMEMory subsystem (see chapter 7). These syntax rules also show how data may be formatted when sent back from the HP 1660 series as a response.

The parameter list always follows the instruction header and is separated from it by white space. When more than one parameter is used, they are separated by commas. You are allowed to include one or more white spaces around the commas, but it is not mandatory.

Numeric data

For numeric data, you have the option of using exponential notation or using suffixes to indicate which unit is being used. However, exponential notation is only applicable to the decimal number base. Tables 5-1 and 5-2 in chapter 5, "Message Communications and System Functions," list all available suffixes. Do not combine an exponent with a unit.

Example The following numbers are all equal:

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

The base of a number is shown with a prefix. The available bases are binary (#B), octal (#Q), hexadecimal (#H) and decimal (default).

Example The following numbers are all equal:

```
#B11100 = #Q34 = #H1C = 28
```

You may not specify a base in conjunction with either exponents or unit suffixes. Additionally, negative numbers must be expressed in decimal.

When a syntax definition specifies that a number is an integer, that means that the number should be whole. Any fractional part would be ignored, truncating the number. Numeric parameters that accept fractional values are called real numbers.

All numbers are expected to be strings of ASCII characters. Thus, when sending the number 9, you send a byte representing the ASCII code for the character "9" (which is 57, or 0011 1001 in binary). A three-digit number, like 102, will take up three bytes (ASCII codes 49, 48 and 50). This is taken care of automatically when you include the entire instruction in a string.

String data

String data may be delimited with either single (') or double (") quotes. String parameters representing labels are case-sensitive. For instance, the labels "Bus A" and "bus a" are unique and should not be used indiscriminately. Also pay attention to the presence of spaces, because they act as legal characters just like any other. So, the labels "In" and " In" are also two different labels.

Keyword data

In many cases a parameter must be a keyword. The available keywords are always included with the instruction's syntax definition. When sending commands, either the longform or shortform (if one exists) may be used. Uppercase and lowercase letters may be mixed freely. When receiving responses, upper-case letters will be used exclusively. The use of longform or shortform in a response depends on the setting you last specified via the SYSTem:LONGform command (see chapter 6).

Selecting Multiple Subsystems

You can send multiple program commands and program queries for different subsystems on the same line by separating each command with a semicolon. The colon following the semicolon enables you to enter a new subsystem.
`<instruction header><data>;:<instruction header><data> <terminator>`

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

Example

```
:MACHINE1:ASSIGN2;:SYSTEM:HEADERS ON
```

Receiving Information from the Instrument

After receiving a query (logic analyzer instruction followed by a question mark), the logic analyzer interrogates the requested function 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 message is transmitted across the bus to the designated listener (typically a controller). The input statement for receiving a response message from an logic analyzer's output queue usually has two parameters: the device address and a format specification for handling the response message.

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 `:MACHINE1:ASSIGN?`, you must follow that query with an input statement. In Basic, this is usually done with an ENTER statement.

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

Example

To read the result of the query command `:SYSTEM:LONGFORM?` you can execute this Basic statement to enter the current setting for the long form command in the numeric variable Setting.

```
ENTER XXX; Setting
```

Response Header Options

The format of the returned ASCII string depends on the current settings of the SYSTEM HEADER and LONGFORM commands. The general format is `<instruction_header><space><data><terminator>`

The header identifies the data that follows (the parameters) and is controlled by issuing a `:SYSTEM:HEADER ON/OFF` command. If the state of the header command is OFF, only the data is returned by the query.

The format of the header is controlled by the :SYSTEM:LONGFORM ON/OFF command. If long form is OFF , the header will be in its short form and the header will vary in length, depending on the particular query. The separator between the header and the data always consists of one space.

A command or query may be sent in either long form or short form, or in any combination of long form and short form. The HEADER and LONGFORM commands only control the format of the returned data, and, they have no affect on the way commands are sent.

Refer to chapter 10, "SYSTem Subsystem" for information on turning the HEADER and LONGFORM commands on and off.

Examples

The following examples show some possible responses for a `:MACHINE1:SFORMAT:THRESHOLD2?` query:

with HEADER OFF:

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

with HEADER ON and LONGFORM OFF:

```
:MACH1:SFOR:THR2 <white_space><data><terminator>
```

with HEADER ON and LONGFORM ON:

```
:MACHINE1:SFORMAT:THRESHOLD2 <white_space><data><terminator>
```

Response Data Formats

Both numbers and strings are returned as a series of ASCII characters, as described in the following sections. Keywords in the data are returned in the same format as the header, as specified by the LONGform command. Like the headers, the keywords will always be in uppercase.

Examples

The following are possible responses to the `MACHINE1:TFORMAT:LAB? 'ADDR' query.`

Header on; Longform on

```
MACHINE1:TFORMAT:LABEL "ADDR  ",19,POSITIVE<terminator>
```

Header on;Longform off

```
MACH1:TFOR:LAB "ADDR  ",19,POS<terminator>
```

Header off; Longform on

```
"ADDR  ",19,POSITIVE<terminator>
```

Header off; Longform off

```
"ADDR  ",19,POS<terminator>
```

Refer to the individual commands in Part 2 of this guide for information on the format (alpha or numeric) of the data returned from each query.

String Variables

Because there are so many ways to code numbers, the HP 1660 series handles almost all data as ASCII strings. Depending on your host language, you may be able to use other types when reading in responses.

Sometimes it is helpful to use string variables in place of constants to send instructions to the HP 1660 series, such as, including the headers with a query response.

Example This example combines variables and constants in order to make it easier to switch from MACHINE1 to MACHINE2. In Basic, the & operator is used for string concatenation.

```
10 LET Machine$ = ":MACHINE2"                    !Send all instructions to machine 2
20 OUTPUT XXX; Machine$ & ":TYPE STATE"          !Make machine a state analyzer
30                                               ! Assign all labels to be positive
40 OUTPUT XXX; Machine$ & ":SFORMAT:LABEL 'CHAN 1', POS"
50 OUTPUT XXX; Machine$ & ":SFORMAT:LABEL 'CHAN 2', POS"
60 OUTPUT XXX; Machine$ & ":SFORMAT:LABEL 'OUT', POS"
99 END
```

If you want to observe the headers for queries, you must bring the returned data into a string variable. Reading queries into string variables requires little attention to formatting.

Example This command line places the output of the query in the string variable Result$.

```
ENTER XXX;Result$
```

In the language used for this book (HP Basic 6.2), string variables are case-sensitive and must be expressed exactly the same each time they are used.

The output of the logic analyzer may be numeric or character data depending on what is queried. Refer to the specific commands, in Part 2 of this guide, for the formats and types of data returned from queries.

Example

The following example shows logic analyzer data being returned to a string variable with headers off:

```
10 OUTPUT XXX;":SYSTEM:HEADER OFF"
20 DIM Rang$[30]
30 OUTPUT XXX;":MACHINE1:TWAVEFORM:RANGE?"
40 ENTER XXX;Rang$
50 PRINT Rang$
60 END
```

After running this program, the controller displays: `+1.00000E-05`

Numeric Base

Most numeric data will be returned in the same base as shown onscreen. When the prefix #B precedes the returned data, the value is in the binary base. Likewise, #Q is the octal base and #H is the hexadecimal base. If no prefix precedes the returned numeric data, then the value is in the decimal base.

Numeric Variables

If your host language can convert from ASCII to a numeric format, then you can use numeric variables. Turning off the response headers will help you avoid accidently trying to convert the header into a number.

Example

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

```
10 OUTPUT XXX;":SYSTEM:HEADER OFF"
20 OUTPUT XXX;":MACHINE1:TWAVEFORM:RANGE?"
30 ENTER XXX;Rang
40 PRINT Rang
50 END
```

This time the format of the number (such as, whether or not exponential notation is used) is dependant upon your host language. In Basic, the output will look like: `1.E-5`

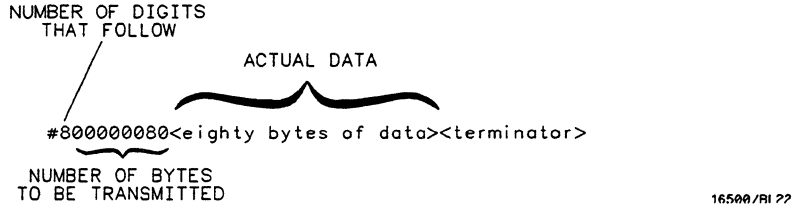
Definite-Length Block Response Data

Definite-length block response data, also refered to as block data, allows any type of device-dependent data to be transmitted over the system interface as a series of data bytes. Definite-length blick data is particularly useful for sending large quantities of data, or, for sending 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. Following the non zero digit is the decimal integer that states the number of 8-bit data bytes to follow. This number is followed by the actual data.

Indefinite-length block data is not supported on the HP1660 series.

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

Figure 1-2



Definite-length Block Response Data

The "8" states the number of digits that follow, and "00000080" states the number of bytes to be transmitted, which is 80.

Multiple Queries

You can send multiple queries to the logic 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.

Example

You can read the result of the query :SYSTEM:HEADER?;LONGFORM? into the string variable Results$ with the command:

```
ENTER XXX; Results$
```

When you read the result of multiple queries into string variables, each response is separated by a semicolon.

Example

The response of the query :SYSTEM:HEADER?:LONGFORM? with HEADER and LONGFORM turned on is:

```
:SYSTEM:HEADER 1;:SYSTEM:LONGFORM 1
```

If you do not need to see the headers when the numeric values are returned, then you could use numeric variables. When you are receiving numeric data into numeric variables, the headers should be turned off. Otherwise the headers may cause misinterpretation of returned data.

Example

The following program message is used to read the query :SYSTEM:HEADERS?;LONGFORM? into multiple numeric variables:

```
ENTER XXX; Result1, Result2
```

Instrument Status

Status registers track the current status of the logic analyzer. By checking the logic analyzer status, you can find out whether an operation has been completed, whether the logic analyzer is receiving triggers, and more. Chapter 6, "Status Reporting," explains how to check the status of the logic analyzer.

2

Programming Over HP-IB

Introduction

This section describes the interface functions and some general concepts of the HP-IB. In general, these functions are defined by IEEE 488.1 (HP-IB bus standard). They deal with general bus management issues, as well as messages which can be sent over the bus as bus commands.

Interface Capabilities

The interface capabilities of the HP 1660-series, as defined by IEEE 488.1 are SH1, AH1, T5, TE0, L3, LE0, SR1, RL1, PP0, DC1, DT1, C0, and E2.

Command and Data Concepts

The HP-IB has two modes of operation: command mode and data mode. The bus is in command mode when the ATN line is true. The command mode is used to send talk and listen addresses and various bus commands, such as a group execute trigger (GET). The bus is in the data mode when the ATN line is false. The data mode is used to convey device-dependent messages across the bus. These device-dependent messages include all of the instrument commands and responses found in chapters 8 through 24 of this manual.

Addressing

By using the front-panel I/O and SELECT keys, the HP-IB interface can be placed in either talk only mode, "Printer connected to HP-IB," or in addressed talk/listen mode, "Controller connected to HP-IB," (see chapter 16, "The RS-232/HP-IB Menu" in the *HP 1660-series User's Reference*). Talk only mode must be used when you want the logic analyzer to talk directly to a printer without the aid of a controller. Addressed talk/listen mode is used when the logic analyzer will operate in conjunction with a controller. When the logic analyzer is in the addressed talk/listen mode, the following is true:

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

If the controller addresses the instrument to talk, it will remain configured to talk until it receives:

- an interface clear message (IFC)
- another instrument's talk address (OTA)
- its own listen address (MLA)
- a universal untalk (UNT) command.

If the controller addresses the instrument to listen, it will remain configured to listen until it receives:

- an interface clear message (IFC)
- its own talk address (MTA)
- a universal unlisten (UNL) command.

Communicating Over the HP-IB Bus (HP 9000 Series 200/300 Controller)

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

Interface Select Code (Selects the Interface)

Each interface card has its own interface select code. This code is used by the controller to direct commands and communications to the proper interface. The default is always "7" for HP-IB controllers.

Instrument Address (Selects the Instrument)

Each instrument on the HP-IB port must have a unique instrument address between decimals 0 and 30. The device address passed with the program message must include not only the correct instrument address, but also the correct interface select code.

Example

For example, if the instrument address is 4 and the interface select code is 7, the instruction will cause an action in the instrument at device address 704.

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

Local, Remote, and Local Lockout

The local, remote, and remote with local lockout modes may be used for various degrees of front-panel control while a program is running. The logic analyzer will accept and execute bus commands while in local mode, and the front panel will also be entirely active. If the HP 1660-series is in remote mode, the logic analyzer will go from remote to local with any front panel activity. In remote with local lockout mode, all controls (except the power switch) are entirely locked out. Local control can only be restored by the controller.

Hint

Cycling the power will also restore local control, but this will also reset certain HP-IB states. It also resets the logic analyzer to the power-on defaults and purges any acquired data in the acquisition memory.

The instrument is placed in remote mode by setting the REN (Remote Enable) bus control line true, and then addressing the instrument to listen. The instrument can be placed in local lockout mode by sending the local lockout (LLO) command (see SYSTem:LOCKout in chapter 9, "Mainframe Commands"). The instrument can be returned to local mode by either setting the REN line false, or sending the instrument the go to local (GTL) command.

Bus Commands

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

Device Clear

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

Group Execute Trigger (GET)

The group execute trigger command will cause the same action as the START command for Group Run: the instrument will acquire data for the active waveform and listing displays.

Interface Clear (IFC)

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

3

Programming Over RS-232C

Introduction

This chapter describes the interface functions and some general concepts of the RS-232C. The RS-232C interface on this instrument is Hewlett-Packard's implementation of EIA Recommended Standard RS-232C, "*Interface Between Data Terminal Equipment and Data Communications Equipment Employing Serial Binary Data Interchange.*" With this interface, data is sent one bit at a time, and characters are not synchronized with preceding or subsequent data characters. Each character is sent as a complete entity without relationship to other events.

Interface Operation

The HP 1660-series can be programmed with a controller over RS-232C using either a minimum three-wire or extended hardwire interface. The operation and exact connections for these interfaces are described in more detail in the following sections. When you are programming an HP 1660-series over RS-232C with a controller, you are normally operating directly between two DTE (Data Terminal Equipment) devices as compared to operating between a DTE device and a DCE (Data Communications Equipment) device.

When operating directly between two DTE devices, certain considerations must be taken into account. For a three-wire operation, XON/XOFF must be used to handle protocol between the devices. For extended hardwire operation, protocol may be handled either with XON/XOFF or by manipulating the CTS and RTS lines of the RS-232C link. For both three-wire and extended hardwire operation, the DCD and DSR inputs to the logic analyzer must remain high for proper operation.

With extended hardwire operation, a high on the CTS input allows the logic analyzer to send data, and a low disables the logic analyzer data transmission. Likewise, a high on the RTS line allows the controller to send data, and a low signals a request for the controller to disable data transmission. Because three-wire operation has no control over the CTS input, internal pull-up resistors in the logic analyzer assure that this line remains high for proper three-wire operation.

Cables

Selecting a cable for the RS-232C interface depends on your specific application. The following paragraphs describe which lines of the HP 1660-series are used to control the operation of the RS-232C relative to the logic analyzer. To locate the proper cable for your application, refer to the reference manual for your controller. This manual should address the exact method your controller uses to operate over the RS-232C bus.

Minimum Three-Wire Interface with Software Protocol

With a three-wire interface, the software (as compared to interface hardware) controls the data flow between the logic analyzer and the controller. The three-wire interface provides no hardware means to control data flow between the controller and the logic analyzer. XON/OFF protocol is the only means to control this data flow. The three-wire interface provides a much simpler connection between devices since you can ignore hardware handshake requirements.

The logic analyzer uses the following connections on its RS-232C interface for three-wire communication:

- Pin 7 SGND (Signal Ground)
- Pin 2 TD (Transmit Data from logic analyzer)
- Pin 3 RD (Receive Data into logic analyzer)

The TD (Transmit Data) line from the logic analyzer must connect to the RD (Receive Data) line on the controller. Likewise, the RD line from the logic analyzer must connect to the TD line on the controller. Internal pull-up resistors in the logic analyzer assure the DCD, DSR, and CTS lines remain high when you are using a three-wire interface.

Extended Interface with Hardware Handshake

With the extended interface, both the software and the hardware can control the data flow between the logic analyzer and the controller. This allows you to have more control of data flow between devices. The HP 1660-series uses the following connections on its RS-232C interface for extended interface communication:

- Pin 7 SGND (Signal Ground)
- Pin 2 TD (Transmit Data from logic analyzer)
- Pin 3 RD (Receive Data into logic analyzer)

The additional lines you use depends on your controller's implementation of the extended hardwire interface.

- Pin 4 RTS (Request To Send) is an output from the logic analyzer which can be used to control incoming data flow.
- Pin 5 CTS (Clear To Send) is an input to the logic analyzer which controls data flow from the logic analyzer.
- Pin 6 DSR (Data Set Ready) is an input to the logic analyzer which controls data flow from the logic analyzer within two bytes.
- Pin 8 DCD (Data Carrier Detect) is an input to the logic analyzer which controls data flow from the logic analyzer within two bytes.
- Pin 20 DTR (Data Terminal Ready) is an output from the logic analyzer which is enabled as long as the logic analyzer is turned on.

The TD (Transmit Data) line from the logic analyzer must connect to the RD (Receive Data) line on the controller. Likewise, the RD line from the logic analyzer must connect to the TD line on the controller.

The RTS (Request To Send), is an output from the logic analyzer which can be used to control incoming data flow. A true on the RTS line allows the controller to send data and a false signals a request for the controller to disable data transmission.

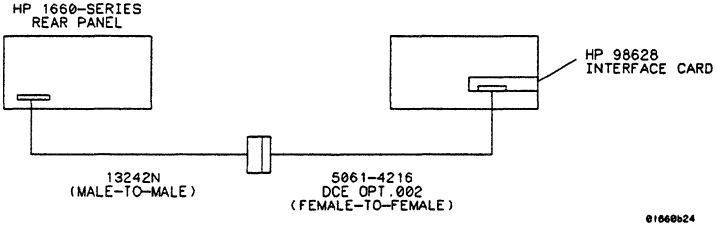
The CTS (Clear To Send), DSR (Data Set Ready), and DCD (Data Carrier Detect) lines are inputs to the logic analyzer, which control data flow from the logic analyzer (Pin 2). Internal pull-up resistors in the logic analyzer assure the DCD and DSR lines remain high when they are not connected. If DCD or DSR are connected to the controller, the controller must keep these lines and the CTS line high to enable the logic analyzer to send data to the controller. A low on any one of these lines will disable the logic analyzer data transmission. Pulling the CTS line low during data transmission will stop logic analyzer data transmission immediately. Pulling either the DSR or DCD line low during data transmission will stop logic analyzer data transmission, but as many as two additional bytes may be transmitted from the logic analyzer.

Cable Example

Figure 3-1 is an example of how to connect the HP 1660-series to the HP 98628A Interface card of an HP 9000 series 200/300 controller. For more information on cabling, refer to the reference manual for your specific controller.

Because this example does not have the correct connections for hardware handshake, you must use the XON/XOFF protocol when connecting the HP 1660-series logic analyzers.

Figure 3-1



Cable Example

Configuring the Instrument Interface

The front-panel I/O menu key allows you access to the RS-232C Configuration menu where the RS-232C interface is configured. If you are not familiar with how to configure the RS-232C interface, refer to the *HP 1660-Series User's Reference*.

Interface Capabilities

The baud rate, stop bits, parity, protocol, and data bits must be configured exactly the same for both the controller and the logic analyzer to properly communicate over the RS-232C bus. The HP 1660-series RS-232C interface capabilities are listed below:

- Baud Rate: 110, 300, 600, 1200, 2400, 4800, 9600, or 19.2k
- Stop Bits: 1, 1.5, or 2
- Parity: None, Odd, or Even
- Protocol: None or XON/XOFF
- Data Bits: 8

Protocol

NONE With a three-wire interface, selecting NONE for the protocol does not allow the sending or receiving device to control data flow. No control over the data flow increases the possibility of missing data or transferring incomplete data.

With an extended hardwire interface, selecting NONE allows a hardware handshake to occur. With hardware handshake, the hardware signals control data flow.

XON/XOFF XON/XOFF stands for Transmit On/Transmit Off. With this mode, the receiver (controller or logic analyzer) controls data flow, and, can request that the sender (logic analyzer or controller) stop data flow. By sending XOFF (ASCII 19) over its transmit data line, the receiver requests that the sender disables data transmission. A subsequent XON (ASCII 17) allows the sending device to resume data transmission.

Data Bits

Data bits are the number of bits sent and received per character that represent the binary code of that character. Characters consist of either 7 or 8 bits, depending on the application. The HP 1660-series supports 8 bit only.

8 Bit Mode Information is usually stored in bytes (8 bits at a time). With 8-bit mode, you can send and receive data just as it is stored, without the need to convert the data.

The controller and the HP 1660-series must be in the same bit mode to properly communicate over the RS-232C. This means that the controller must have the capability to send and receive 8 bit data.

For more information on the RS-232C interface, refer to the *HP 1660-series User's Reference*. For information on RS-232C voltage levels and connector pinouts, refer to the *HP 1660-Series Service Guide*.

Communicating Over the RS-232C Bus (HP 9000 Series 200/300 Controller)

Each RS-232C interface card has its own interface select code. This code is used by the controller for directing commands and communications to the proper interface by specifying the correct interface code for the device address.

Generally, the interface select code can be any decimal value between 0 and 31, except for those interface codes which are reserved by the controller for internal peripherals and other internal interfaces. This value can be selected through switches on the interface card. For example, if your RS-232C interface select code is 9, the device address required to communicate over the RS-232C bus is 9. For more information, refer to the reference manual for your interface card or controller.

Lockout Command

To lockout the front panel controls, use the SYSTem command LOCKout. When this function is on, all controls (except the power switch) are entirely locked out. Local control can only be restored by sending the command :LOCKout OFF.

Hint

Cycling the power will also restore local control, but this will also reset certain RS-232C states. It also resets the logic analyzer to the power-on defaults and purges any acquired data in the acquisition memory.

For more information on this command see chapter 10, "System Commands."

4

Programming and Documentation Conventions

Introduction

This chapter covers the programming conventions used in programming the instrument, as well as the documentation conventions used in this manual. This chapter also contains a detailed description of the command tree and command tree traversal.

Truncation Rule

The truncation rule for the keywords used in headers and parameters is:

- If the longform has four or fewer characters, there is no change in the shortform. When the longform has more than four characters the shortform is just the first four characters, unless the fourth character is a vowel. In that case only the first three characters are used.

> There are some commands that do not conform to the truncation rule by design. These will be noted in their respective description pages.

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

Table 4-1 Truncation Examples

Long Form	Short Form
OFF	OFF
DATA	DATA
START	STAR
LONGFORM	LONG
DELAY	DEL
ACCUMULATE	ACC

Infinity Representation

The representation of infinity is 9.9E+37 for real numbers and 32767 for integers. This is also the value returned when a measurement cannot be made.

Sequential and Overlapped Commands

IEEE 488.2 makes the distinction between sequential and overlapped commands. Sequential commands finish their task before the execution of the next command starts. Overlapped commands run concurrently; therefore, the command following an overlapped command may be started before the overlapped command is completed. The overlapped commands for the HP 1660-series are STARt and STOP.

Response Generation

IEEE 488.2 defines two times at which query responses may be buffered. The first is when the query is parsed by the instrument and the second is when the controller addresses the instrument to talk so that it may read the response. The HP 1660-series will buffer responses to a query when it is parsed.

Syntax Diagrams

At the beginning of each chapter in Part 2, "Commands," is a syntax diagram showing the proper syntax for each command. All characters contained in a circle or oblong are literals, and must be entered exactly as shown. Words and phrases contained in rectangles are names of items used with the command and are described in the accompanying text of each command. Each line can only be entered from one direction as indicated by the arrow on the entry line. Any combination of commands and arguments that can be generated by following the lines in the proper direction is syntactically correct. An argument is optional if there is a path around it. When there is a rectangle which contains the word "space," a white space character must be entered. White space is optional in many other places.

Notation Conventions and Definitions

The following conventions are used in this manual when describing programming rules and example.

< >	Angular brackets enclose words or characters that are used to symbolize a program code parameter or a bus command
::=	"is defined as." For example, A ::= B indicates that A can be replaced by B in any statement containing A.
|	"or." Indicates a choice of one element from a list. For example, A | B indicates A or B, but not both.
. . .	An ellipsis (trailing dots) is used to indicate that the preceding element may be repeated one or more times.
[ ]	Square brackets indicate that the enclosed items are optional.
{ }	When several items are enclosed by braces and separated by vertical bars (|), one, and only one of these elements must be selected.
XXX	Three Xs after an ENTER or OUTPUT statement represent the device address required by your controller.
<NL>	Linefeed (ASCII decimal 10).

The Command Tree

The command tree (figure 4-1) shows all commands in the HP 1660-series logic analyzers and the relationship of the commands to each other. Parameters are not shown in this figure. The command tree allows you to see what the HP 1660-series' parser expects to receive. All legal headers can be created by traversing down the tree, adding keywords until the end of a branch has been reached.

Command Types

As shown in chapter 1, "Header Types," there are three types of headers. Each header has a corresponding command type. This section shows how they relate to the command tree.

System Commands The system commands reside at the top level of the command tree. These commands are always parsable if they occur at the beginning of a program message, or are preceded by a colon. START and STOP are examples of system commands.

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

Common Commands Common commands are independent of the tree, and do not affect the position of the parser within the tree. *CLS and *RST are examples of common commands.

Tree Traversal Rules

Command headers are created by traversing down the command tree. For each group of keywords not separated by a branch, one keyword must be selected. As shown on the tree, branches are always preceded by colons. Do not add spaces around the colons. The following two rules apply to traversing the tree:

A leading colon (the first character of a header) or a <terminator> places the parser at the root of the command tree.

Executing a subsystem command places you in that subsystem until a leading colon or a <terminator> is found. The parser will stay at the colon above the keyword where the last header terminated. Any command below that point can be sent within the current program message without sending the keywords(s) which appear above them.

The following examples are written using HP Basic 6.2 on a HP 9000 Series 200/300 Controller. The quoted string is placed on the bus, followed by a carriage return and linefeed (CRLF). The three Xs (XXX) shown in this manual after an ENTER or OUTPUT statement represents the device address required by your controller.

Example 1

In this example, the colon between **SYSTEM** and **HEADER** is necessary since **SYSTEM:HEADER** is a compound command. The semicolon between the **HEADER** command and the **LONGFORM** command is the required **<program message unit separator>** . The **LONGFORM** command does not need **SYSTEM** preceding it, since the **SYSTEM:HEADER** command sets the parser to the **SYSTEM** node in the tree.

```
OUTPUT XXX;":SYSTEM:HEADER ON;LONGFORM ON"
```

Example 2

In the first line of this example, the subsystem selector is implied for the **STORE** command in the compound command. The **STORE** command must be in the same program message as the **INITIALIZE** command, since the **<program message terminator>** will place the parser back at the root of the command tree.

A second way to send these commands is by placing **MMEMORY:** before the **STORE** command as shown in the fourth line of this example 2.

```
OUTPUT XXX;":MMEMORY:INITIALIZE;STORE 'FILE  ','FILE
DESCRIPTION'"
```

or

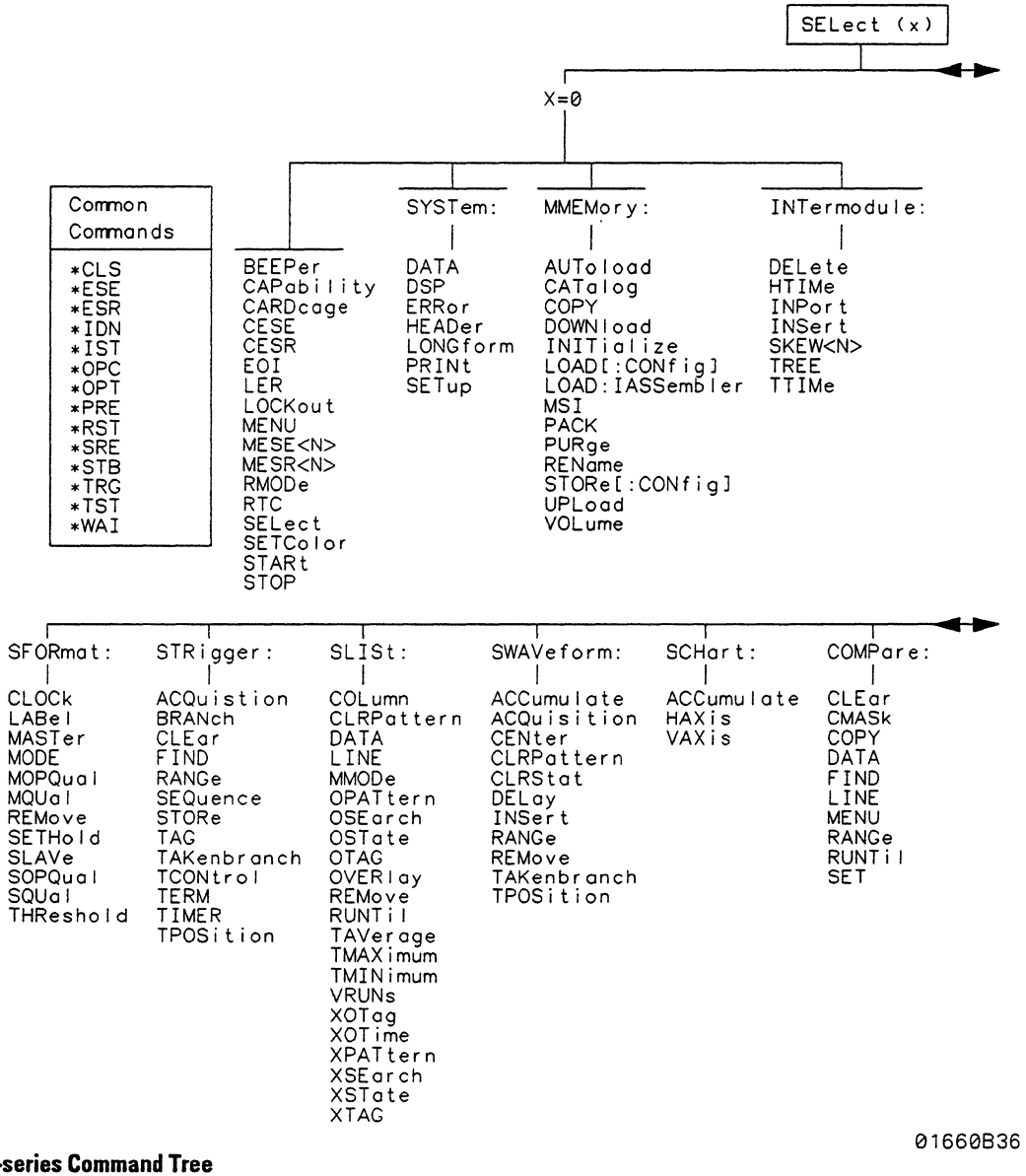
```
OUTPUT XXX;":MMEMORY:INITIALIZE"
OUTPUT XXX;":MMEMORY:STORE 'FILE  ','FILE DESCRIPTION'"
```

Example 3

In this example, the leading colon before **SYSTEM** tells the parser to go back to the root of the command tree. The parser can then see the **SYSTEM:PRINT** command.

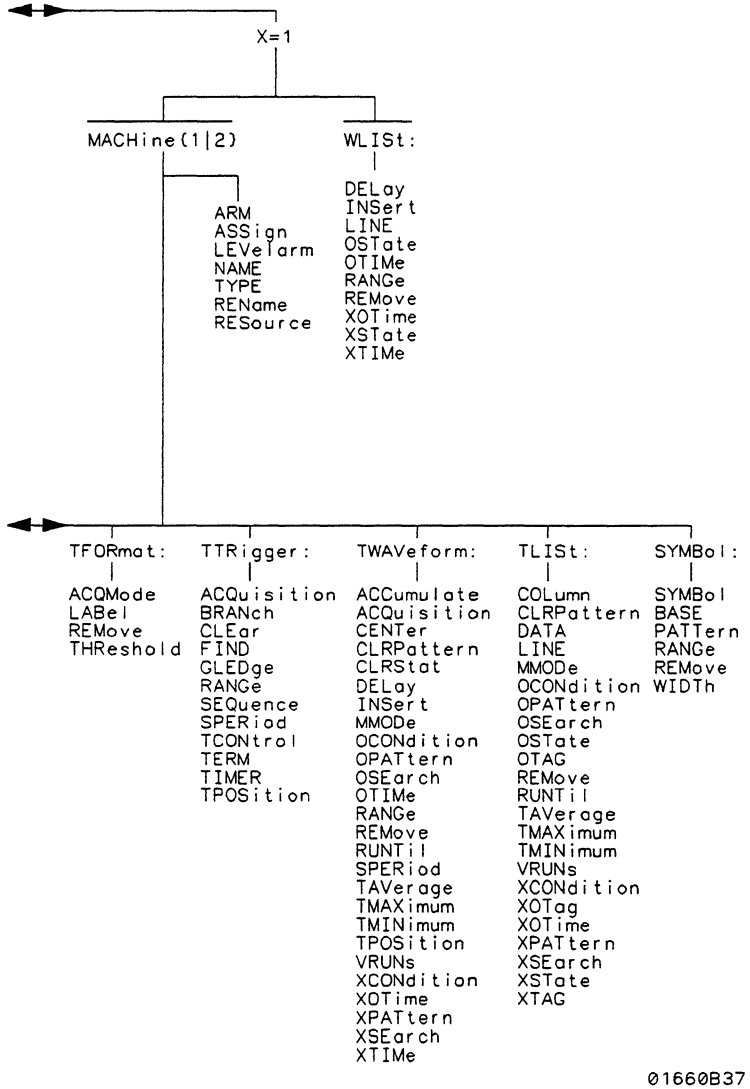
```
OUTPUT XXX;":MMEM:CATALOG?;:SYSTEM:PRINT ALL"
```

Figure 4-1



HP 1660-series Command Tree

Figure 4-1 (continued)



HP 1660-series Command Tree (continued)

Table 4-2

Alphabetic Command Cross-Reference

Command	Subsystem
ACCumulate	SCHart, SWAVeform, TWAVeform
ACQMode	TFORmat
ACQuisition	STRigger, SWAVeform, TTRigger, TWAVeform
ARM	MACHine
ASSign	MACHine
AUToload	MMEMory
BASE	SYMBol
BEEPer	Mainframe
BRANch	STRigger, TTRigger
CAPability	Mainframe
CARDcage	Mainframe
CATalog	MMEMory
CENTer	SWAVeform, TWAVeform
CESE	Mainframe
CESR	Mainframe
CLEar	COMPare, STRigger, TTRigger
CLOCk	SFORmat
CLRPattern	SLISt, SWAVeform, TLISt, TWAVeform
CLRStat	SWAVeform, TWAVeform
CMASk	COMPare
COLumn	SLISt, TLISt
COPY	COMPare, MMEMory
DATA	COMPare, SLISt, SYSTem, TLISt
DELay	SWAVeform, TWAVeform, WLISt
DELete	INTermodule
DOWNload	MMEMory
DSP	SYSTem
EOI	Mainframe
ERRor	SYSTem
FIND	COMPare, STRigger, TTRigger
GLEDge	TTRigger
HAXis	SCHart
HEADer	SYSTem
HTIMe	INTermodule
INITialize	MMEMory
INPort	INTermodule
INSert	INTermodule, SWAVeform, TWAVeform, WLISt
LABel	SFORmat, TFORmat
LER	Mainframe
LEVelarm	MACHine
LINE	COMPare, SLISt, TLISt, WLISt
LOAD	MMEMory
LOCKout	Mainframe
LONGform	SYSTem
MASTer	SFORmat
MENU	COMPare, Mainframe
MESE	Mainframe
MESR	Mainframe

Table 4-2 (continued)

Alphabetic Command Cross-Reference (continued)

Command	Subsystem	Command	Subsystem
MMODe	SLISt, TLISt, TWAVeform	RUNTil	COMPare, SLISt, TLISt, TWAVeform
MODE	SFORmat	SELect	Mainframe
MOPQual	SFORmat	SEQuence	STRigger, TTRigger
MQUal	SFORmat	SET	COMPare
MSI	MMEMory	SETColor	Mainframe
NAME	MACHine	SETHold	SFORmat
OCONdition	TLISt, TWAVeform	SETup	SYSTem
OPATtern	SLISt, TLISt, TWAVeform	SKEW	INTermodule
OSEarch	SLISt, TLISt, TWAVeform	SLAVe	SFORmat
OSTate	SLISt, TLISt, WLISt	SOPQual	SFORmat
OTAG	SLISt, TLISt	SPERiod	TTRigger, TWAVeform
OTIMe	TWAVeform, WLISt	SQUal	SFORmat
OVERlay	SLISt	STARt	Mainframe
PACK	MMEMory	STOP	Mainframe
PATTern	SYMBol	STORe	MMEMory, STRigger
PRINt	SYSTem	TAG	STRigger
PURGe	MMEMory	TAKenbranch	STRigger, SWAVeform
RANGe	COMPare, STRigger, SWAVeform, SYMBol, TTRigger, TWAVeform, WLISt	TAVerage	SLISt, TLISt, TWAVeform
REMove	SFORmat, SLISt, SWAVeform, SYMBol, TFORmat, TLISt, TWAVeform,	TCONtrol	STRigger, TTRigger
REName	MACHine	TERM	STRigger, TTRigger
REName	MMEMory	THReshold	SFORmat, TFORmat
RESource	MACHine	TIMER	STRigger, TTRigger
RMODe	Mainframe	TMAXimum	SLISt, TLISt, TWAVeform
RTC	Mainframe	TMINimum	SLISt, TLISt, TWAVeform
		TPOSition	STRigger, SWAVeform, TTRigger, TWAVeform

Table 4-2 (continued)

Alphabetic Command Cross-Reference (continued)

Command	Subsystem	Command	Subsystem
TREE	Intermodule	WIDTh	SYMBol
TTIMe	INTermodule	XCONdition	TLISt, TWAVeform
TYPE	MACHine	XOTag	SLISt, TLISt
UPLoad	MMEMory	XOTime	SLISt, TLISt, TWAVeform, WLISt
VAXis	SCHart	XPATtern	SLISt, TLISt, TWAVeform
VOLume	MMEMory	XSEarch	SLISt, TLISt, TWAVeform
VRUNs	SLISt, TLISt, TWAVeform	XSTate	SLISt, TLISt, WLISt

Command Set Organization

The command set for the HP 1660-series logic analyzers is divided into 18 separate groups: common commands, mainframe commands, system commands and 15 sets of subsystem commands. Each of the 18 groups of commands is described in a seperate chapter in Part 2, "Commands." Each of the chapters contain a brief description of the subsystem, a set of syntax diagrams for those commands, and finally, the commands for that subsystem in alphabetical order. The commands are shown in the long form and short form using upper and lowercase letters. As an example **AUToload** indicates that the long form of the command is **AUTOLOAD** and the short form of the command is **AUT**. Each of the commands contain a description of the command, its arguments, and the command syntax.

Subsystems

There are 15 subsystems in this instrument. In the command tree (figure 4-1) they are shown as branches, with the node above showing the name of the subsystem. Only one subsystem may be selected at a time. At power on, the command parser is set to the root of the command tree; therefore, no subsystem is selected. The 15 subsystems in the HP 1660-series logic analyzers are:

- SYSTem - controls some basic functions of the instrument.
- MMEMory - provides access to the internal disk drive.
- INTermodule - provides access to the Intermodule bus (IMB).
- MACHine - provides access to analyzer functions and subsystems.
- WLISt - allows access to the mixed (timing/state) functions.
- SFORmat - allows access to the state format functions.
- STRigger - allows access to the state trigger functions.
- SLISt - allows access to the state listing functions.
- SWAVeform - allows access to the state waveforms functions.
- SCHart - allows access to the state chart functions.
- COMPare - allows access to the compare functions.
- TFORmat - allows access to the timing format functions.
- TTRigger - allows access to the timing trigger functions.
- TWAVeform - allows access to the timing waveforms functions.
- TLISt - allows access to the timing listing functions.
- SYMBol - allows access to the symbol specification functions.

Program Examples

The program examples in the following chapters and chapter 27, "Programming Examples," were written on an HP 9000 Series 200/300 controller using the HP Basic 6.2 language. The programs always assume a generic address for the HP 1660-series logic analyzers of XXX.

In the examples, you should pay special attention to the ways in which the command and/or query can be sent. Keywords can be sent using either the long form or short form (if one exists for that word). With the exception of some string parameters, the parser is not case-sensitive. Uppercase and lowercase letters may be mixed freely. System commands like HEADer and LONGform allow you to dictate what forms the responses take, but they have no affect on how you must structure your commands and queries.

Example

The following commands all set the Timing Waveform Delay to 100 ms.

Keywords in long form, numbers using the decimal format.

```
OUTPUT XXX;":MACHINE1:TWAVEFORM:DELAY .1"
```

Keywords in short form, numbers using an exponential format.

```
OUTPUT XXX;":MACH1:TWAV:DEL 1E-1"
```

Keywords in short form using lowercase letters, numbers using a suffix.

```
OUTPUT XXX;":mach1:twav:del 100ms"
```

In these examples, the colon shown as the first character of the command is optional on the HP 1660-series. The space between DELay and the argument is required.

5

Message Communication and System Functions

Introduction

This chapter describes the operation of instruments 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 logic 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."*

The HP 1660 series is designed to be compatible with other Hewlett-Packard IEEE 488.2 compatible instruments. Instruments that are compatible with IEEE 488.2 must also be compatible with IEEE 488.1 (HP-IB bus standard); however, IEEE 488.1 compatible instruments may or may not conform to the IEEE 488.2 standard. The IEEE 488.2 standard defines the message exchange protocols by which the instrument and the controller will communicate. It also defines some common capabilities, which are found in all IEEE 488.2 instruments. This chapter also contains a few items which are not specifically defined by IEEE 488.2, but deal with message communication or system functions.

The syntax and protocol for RS-232C program messages and response messages for the HP 1660-series are structured very similar to those described by 488.2. In most cases, the same structure shown in this chapter for 488.2 will also work for RS-232C. Because of this, no additional information has been included for RS-232C.

Protocols

The protocols of IEEE 488.2 define the overall scheme used by the controller and the instrument to communicate. This includes defining when it is appropriate for devices to talk or listen, and what happens when the protocol is not followed.

Functional Elements

Before proceeding with the description of the protocol, a few system components should be understood.

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

Output Queue The output queue of the instrument is the memory area where all output data (<response messages>) are stored until read by the controller.

Parser The instrument's parser is the component that interprets the commands sent to the instrument and decides what actions should be taken. "Parsing" refers to the action taken by the parser to achieve this goal. Parsing and executing of commands begins when either the instrument recognizes a <program message terminator> (defined later in this chapter) or the input buffer becomes full. If you wish to send a long sequence of commands to be executed and then talk to another instrument while they are executing, you should send all the commands before sending the <program message terminator>.

Protocol Overview

The instrument and controller communicate using <program message>s and <response message>s. These messages serve as the containers into which sets of program commands or instrument responses are placed. <program message>s are sent by the controller to the instrument, and <response message>s are sent from the instrument to the controller in response to a query message. A <query message> is defined as being a <program message> which contains one or more queries. The instrument will only talk when it has received a valid query message, and therefore has something to say. The controller should only attempt to read a response after sending a complete query message, but before sending another <program message>. The basic rule to remember is that the instrument will only talk when prompted to, and it then expects to talk before being told to do something else.

Protocol Operation

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

The instrument and the controller communicate by exchanging complete <program message>s and <response message>s. This means that the controller should always terminate a <program message> before attempting to read a response. The instrument will terminate <response message>s except during a hardcopy output.

If a query message is sent, the next message passing over the bus should be the <response message>. The controller should always read the complete <response message> associated with a query message before sending another <program message> to the same instrument.

The instrument allows the controller to send multiple queries in one query message. This is referred to as sending a "compound query." As will be noted later in this chapter, 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. Some of the protocol exceptions are shown below.

Command Error A command error will be reported if the instrument detects a syntax error or an unrecognized command header.

Execution Error An execution error will be reported if a parameter is found to be out of range, or if the current settings do not allow execution of a requested command or query.

Device-specific Error A device-specific error will be reported if the instrument is unable to execute a command for a strictly device dependent reason.

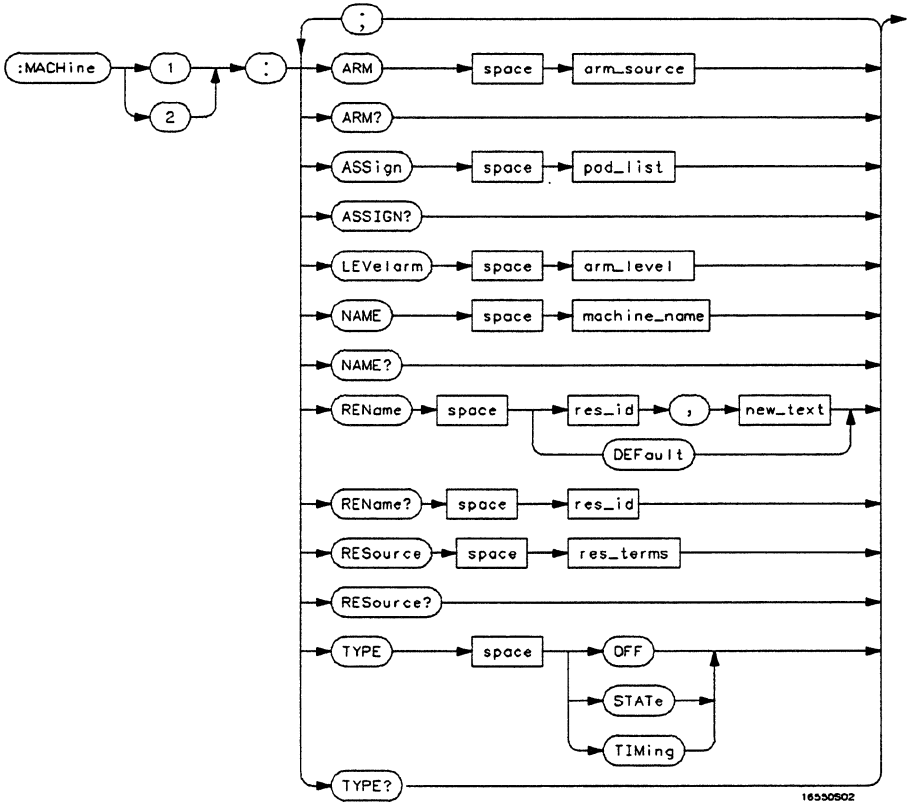
Query Error A query error will be reported if the proper protocol for reading a query is not followed. This includes the interrupted and unterminated conditions described in the following paragraphs.

Syntax Diagrams

The example syntax diagram is in this chapter are similar to the syntax diagrams in the IEEE 488.2 specification. Commands and queries are sent to the instrument as a sequence of data bytes. The allowable byte sequence for each functional element is defined by the syntax diagram that is shown.

The allowable byte sequence can be determined by following a path in the syntax diagram. The proper path through the syntax diagram is any path that follows the direction of the arrows. If there is a path around an element, that element is optional. If there is a path from right to left around one or more elements, that element or those elements may be repeated as many times as desired.

Figure 5-1



Example syntax diagram

Syntax Overview

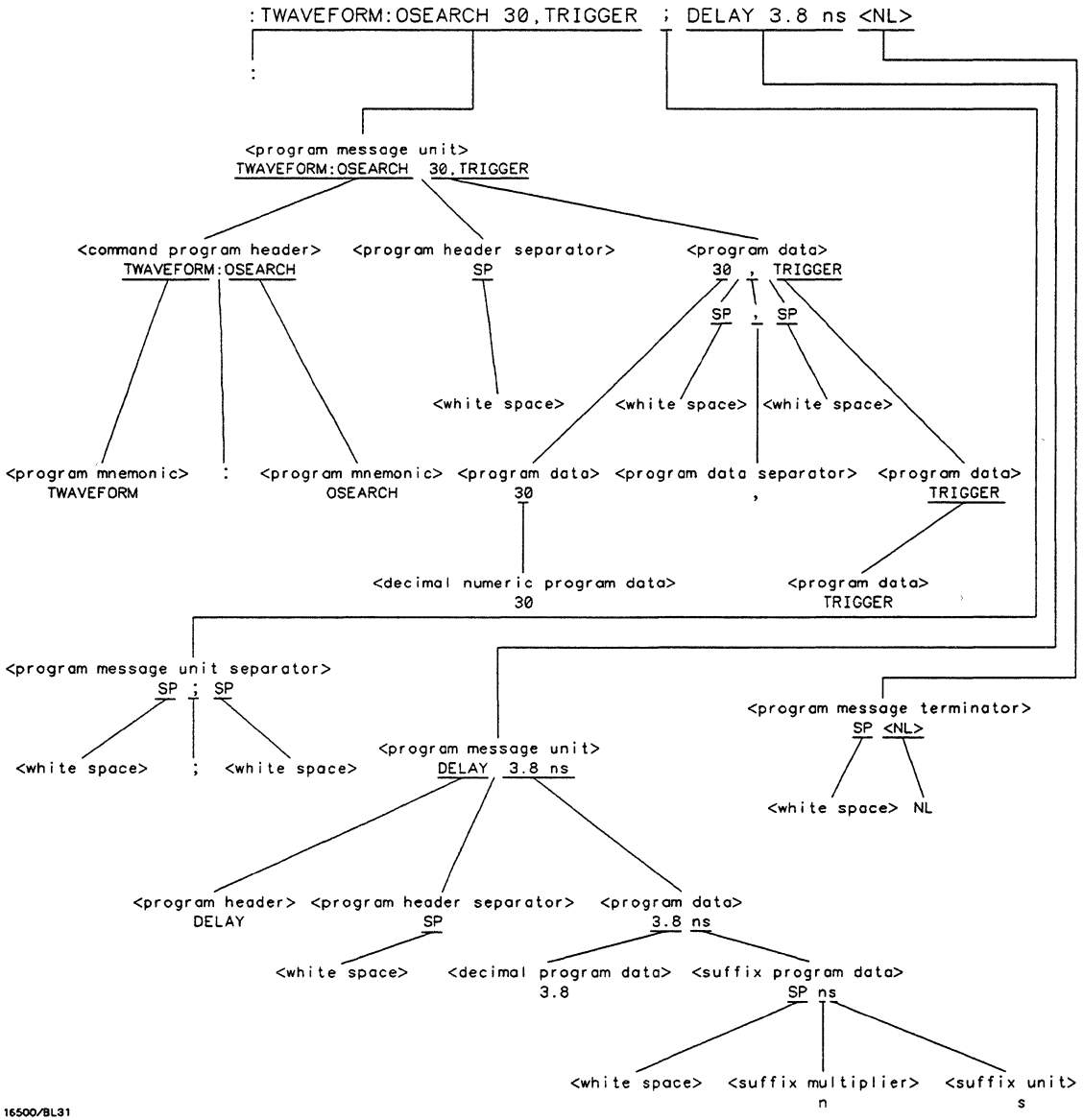
This overview is intended to give a quick glance at the syntax defined by IEEE 488.2. It will help you understand many of the things about the syntax you need to know.

IEEE 488.2 defines the blocks used to build messages which are sent to the instrument. A whole string of commands can therefore be broken up into individual components.

Figure 5-1 is an example syntax diagram and figure 5-2 shows a breakdown of an example <program message>. There are a few key items to notice:

- A semicolon separates commands from one another. Each <program message unit> serves as a container for one command. The <program message unit>s are separated by a semicolon.
- A <program message> is terminated by a <NL> (new line). The recognition of the <program message terminator>, or <PMT>, by the parser serves as a signal for the parser to begin execution of commands. The <PMT> also affects command tree traversal (Chapter 4, "Programming and Documentation Conventions").
- Multiple data parameters are separated by a comma.
- The first data parameter is separated from the header with one or more spaces.
- The header MACHINE1:ASSIGN 2,3 is an example of a compound header. It places the parser in the machine subsystem until the <NL> is encountered.
- A colon preceding the command header returns you to the top of the command tree.

Figure 5-2



16500/BL31

<program message> Parse Tree

Upper/Lower Case Equivalence

Upper and lower case letters are equivalent. The mnemonic `SINGLE` has the same semantic meaning as the mnemonic `single`.

<white space>

<white space> is defined to be one or more characters from the ASCII set of 0 - 32 decimal, excluding 10 decimal (NL). <white space> is used by several instrument listening components of the syntax. It is usually optional, and can be used to increase the readability of a program.

Suffix Multiplier The suffix multipliers that the instrument will accept are shown in table 5-1.

Table 5-1 **<suffix mult>**

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

Suffix Unit The suffix units that the instrument will accept are shown in table 5-2.

Table 5-2

<suffix unit>

Suffix	Referenced Unit
V	Volt
S	Second

6

Status Reporting

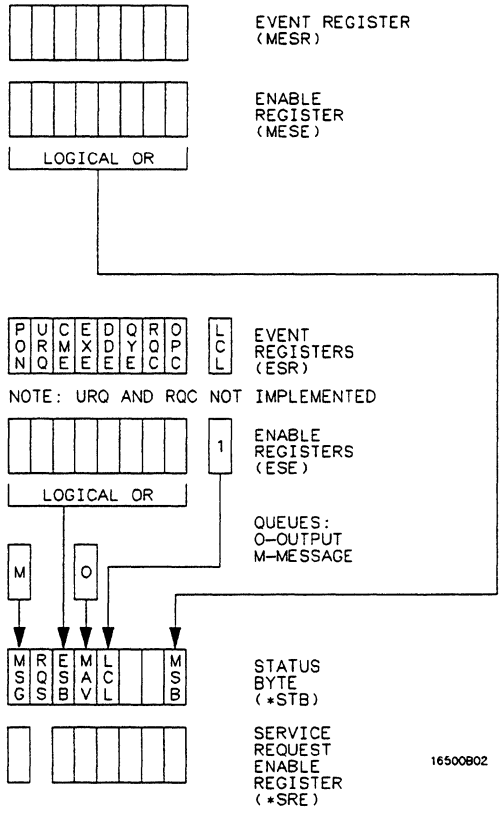
Introduction

Status reporting allows you to use information about the instrument in your programs, so that you have better control of the measurement process. For example, you can use status reporting to determine when a measurement is complete, thus controlling your program, so that it does not get ahead of the instrument. This chapter describes the status registers, status bytes and status bits defined by IEEE 488.2 and discusses how they are implemented in the HP 1660-series logic analyzers. Also in this chapter is a sample set of steps you use to perform a serial poll over HP-IB.

The status reporting feature available over the bus is the serial poll. IEEE 488.2 defines data structures, commands, and common bit definitions. There are also instrument-defined structures and bits.

The bits in the status byte act as summary bits for the data structures residing behind them. In the case of queues, the summary bit is set if the queue is not empty. For registers, the summary bit is set if any enabled bit in the event register is set. The events are enabled via the corresponding event enable register. Events captured by an event register remain set until the register is read or cleared. Registers are read with their associated commands. The *CLS 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 will also be cleared.

Figure 6-1



Status Byte Structures and Concepts

Event Status Register

The Event Status Register is an IEEE 488.2 defined register. The bits in this register are "latched." That is, once an event happens which sets a bit, that bit will only be cleared if the register is read.

Service Request Enable Register

The Service Request Enable Register is an 8-bit register. Each bit enables the corresponding bit in the status byte to cause a service request. The sixth bit does not logically exist and is always returned as a zero. To read and write to this register, use the *SRE? and *SRE commands.

Bit Definitions

The following mnemonics are used in figure 6-1 and in chapter 8, "Common Commands:"

MAV - message available

Indicates whether there is a response in the output queue.

ESB - event status bit

Indicates if any of the conditions in the Standard Event Status Register are set and enabled.

MSS - master summary status

Indicates whether the device has a reason for requesting service. This bit is returned for the *STB? query.

RQS - request service

Indicates if the device is requesting service. This bit is returned during a serial poll. RQS will be set to 0 after being read via a serial poll (MSS is not reset by *STB?).

MSG - message

Indicates whether there is a message in the message queue (Not implemented in the HP 1660-series).

PON - power on

Indicates power has been turned on.

URQ - user request

Always returns a 0 from the HP 1660-series.

CME - command error

Indicates whether the parser detected an error.

> The error numbers and strings for CME, EXE, DDE, and QYE can be read from a device-defined queue (which is not part of IEEE 488.2) with the query :SYSTEM:ERROR?.

EXE - execution error

Indicates whether a parameter was out of range, or inconsistent with current settings.

DDE - device specific error

Indicates whether the device was unable to complete an operation for device dependent reasons.

QYE - query error

Indicates whether the protocol for queries has been violated.

RQC - request control

Always returns a 0 from the HP 1660-series.

OPC - operation complete

Indicates whether the device has completed all pending operations. OPC is controlled by the *OPC common command. Because this command can appear after any other command, it serves as a general-purpose operation complete message generator.

LCL - remote to local

Indicates whether a remote to local transition has occurred.

MSB - module summary bit

Indicates that an enable event in one of the modules Status registers has occurred.

Key Features

A few of the most important features of Status Reporting are listed in the following paragraphs.

Operation Complete

The IEEE 488.2 structure provides one technique that can be used to find out if any operation is finished. The *OPC command, when sent to the instrument after the operation of interest, will set the OPC bit in the Standard Event Status Register. If the OPC bit and the RQS bit have been enabled, a service request will be generated. The commands that affect the OPC bit are the overlapped commands.

Example

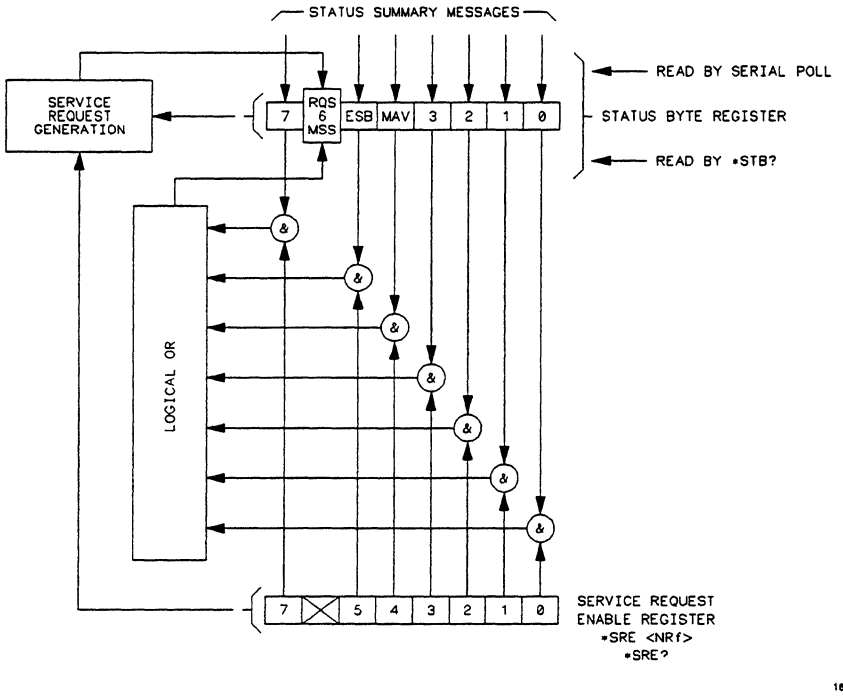
```
OUTPUT XXX;"*SRE 32 ; *ESE 1"  !enables an OPC service request
```

Status Byte

The Status Byte contains the basic status information which is sent over the bus in a serial poll. If the device is requesting service (RQS set), and the controller serial-polls the device, the RQS bit is cleared. The MSS (Master Summary Status) bit (read with *STB?) and other bits of the Status Byte are not be cleared by reading them. Only the RQS bit is cleared when read.

The Status Byte is cleared with the *CLS common command.

Figure 6-2.



Service Request Enabling

Serial Poll

The HP 1660-series supports the IEEE 488.1 serial poll feature. When a serial poll of the instrument is requested, the RQS bit is returned on bit 6 of the status byte.

Using Serial Poll (HP-IB)

This example will show how to use the service request by conducting a serial poll of all instruments on the HP-IB bus. In this example, assume that there are two instruments on the bus: a Logic Analyzer at address 7 and a printer at address 1.

The program command for serial poll using HP BASIC 6.2 is Stat = SPOLL(707). The address 707 is the address of the logic analyzer in the this example. The command for checking the printer is Stat = SPOLL(701) because the address of that instrument is 01 on bus address 7. This command reads the contents of the HP-IB Status Register into the variable called Stat. At that time bit 6 of the variable Stat can be tested to see if it is set (bit 6 = 1).

The serial poll operation can be conducted in the following manner:

1. Enable interrupts on the bus. This allows the controller to see the SRQ line.
2. Disable interrupts on the bus.
3. If the SRQ line is high (some instrument is requesting service) then check the instrument at address 1 to see if bit 6 of its status register is high.
4. To check whether bit 6 of an instruments status register is high, use the following BASIC statement: `IF BIT (Stat, 6) THEN`
5. If bit 6 of the instrument at address 1 is not high, then check the instrument at address 7 to see if bit 6 of its status register is high.
6. As soon as the instrument with status bit 6 high is found check the rest of the status bits to determine what is required.

The SPOLL(707) command causes much more to happen on the bus than simply reading the register. This command clears the bus automatically, addresses the talker and listener, sends SPE (serial poll enable) and SPD (serial poll disable) bus commands, and reads the data. For more information about serial poll, refer to your controller manual, and programming language reference manuals.

After the serial poll is completed, the RQS bit in the HP 1660-series Status Byte Register will be reset if it was set. Once a bit in the Status Byte Register is set, it will remain set until the status is cleared with a *CLS command, or the instrument is reset.

7

Error Messages

Introduction

This chapter lists the error messages that relate to the HP 1660-series Logic Analyzers.

Device Dependent Errors

200 Label not found
201 Pattern string invalid
202 Qualifier invalid
203 Data not available
300 RS-232C error

Command Errors

-100 Command error (unknown command) (generic error)
-101 Invalid character received
-110 Command header error
-111 Header delimiter error
-120 Numeric argument error
-121 Wrong data type (numeric expected)
-123 Numeric overflow
-129 Missing numeric argument
-130 Non numeric argument error (character,string, or block)
-131 Wrong data type (character expected)
-132 Wrong data type (string expected)
-133 Wrong data type (block type #D required)
-134 Data overflow (string or block too long)
-139 Missing non numeric argument
-142 Too many arguments
-143 Argument delimiter error
-144 Invalid message unit delimiter

Execution Errors

-200 Can Not Do (generic execution error)
-201 Not executable in Local Mode
-202 Settings lost due to return-to-local or power on
-203 Trigger ignored
-211 Legal command, but settings conflict
-212 Argument out of range
-221 Busy doing something else
-222 Insufficient capability or configuration
-232 Output buffer full or overflow
-240 Mass Memory error (generic)
-241 Mass storage device not present
-242 No media
-243 Bad media
-244 Media full
-245 Directory full
-246 File name not found
-247 Duplicate file name
-248 Media protected

Internal Errors

-300 Device Failure (generic hardware error)
-301 Interrupt fault
-302 System Error
-303 Time out
-310 RAM error
-311 RAM failure (hardware error)
-312 RAM data loss (software error)
-313 Calibration data loss
-320 ROM error

-321 ROM checksum
-322 Hardware and Firmware incompatible
-330 Power on test failed
-340 Self Test failed
-350 Too Many Errors (Error queue overflow)

Query Errors

-400 Query Error (generic)
-410 Query INTERRUPTED
-420 Query UNTERMINATED
-421 Query received. Indefinite block response in progress
-422 Addressed to Talk, Nothing to Say
-430 Query DEADLOCKED

8

Common Commands

Introduction

The common commands are defined by the IEEE 488.2 standard. These commands must be supported by all instruments that comply with this standard. Refer to figure 8-1 and table 8-1 for the common commands syntax diagram.

The common commands control some of the basic instrument functions; such as, instrument identification and reset, how status is read and cleared, and how commands and queries are received and processed by the instrument.

Common commands can be received and processed by the HP 1660-series logic analyzers, whether they are sent over the bus as separate program messages or within other program messages. If an instrument subsystem has been selected and a common command is received by the instrument, the logic analyzer will remain in the selected subsystem.

Example

If the program message in this example is received by the logic analyzer, it will initialize the disk and store the file and clear the status information. This is not be the case if some other type of command is received within the program message.

```
":MMEMORY:INITIALIZE;*CLS; STORE 'FILE  ','DESCRIPTION'"
```

Example

This program message initializes the disk, selects the module in slot A, then stores the file. In this example, :MMEMORY must be sent again in order to reenter the memory subsystem and store the file.

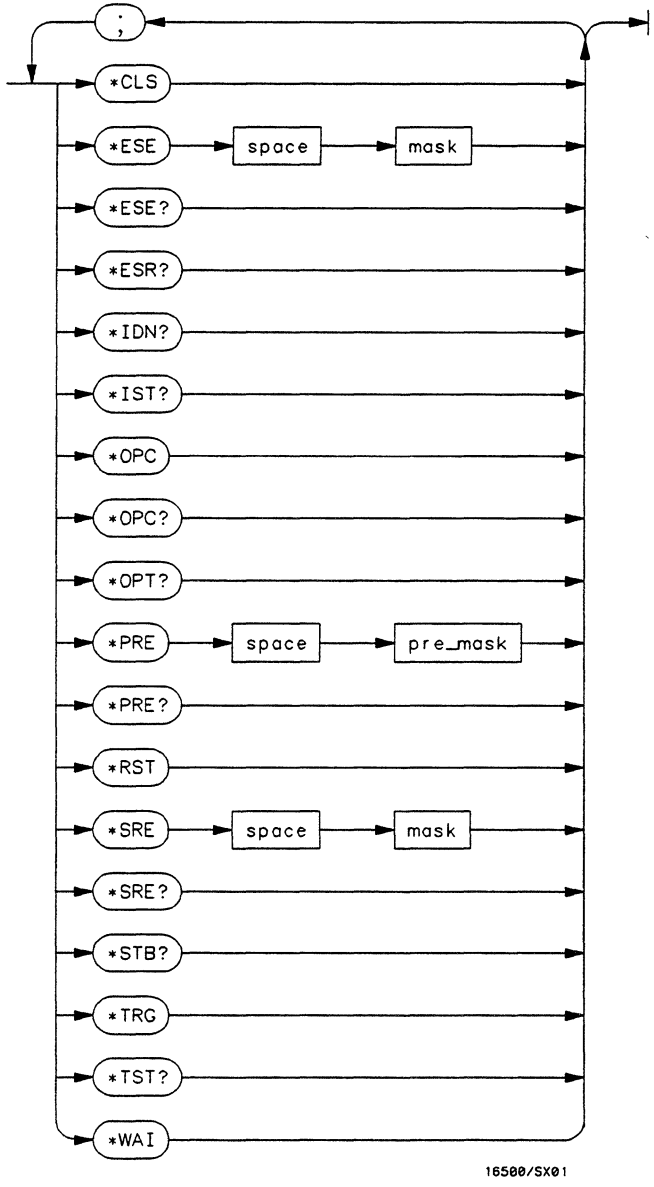
```
":MMEMORY:INITIALIZE;:SELECT 1;:MMEMORY:STORE 'FILE  ',
'DESCRIPTION'"
```

Status Registers

Each status register has an associated status enable (mask) register. By setting the bits in the status enable register you can select the status information you wish to use. Any status bits that have not been masked (enabled in the enable register) will not be used to report status summary information to bits in other status registers.

Refer to chapter 6, "Status Reporting," for a complete discussion of how to read the status registers and how to use the status information available from this instrument.

Figure 8-1



Common Commands Syntax Diagram

Table 8-1 **Common Command Parameter Values**

Parameter	Values
mask	An integer, 0 through 255.
pre_mask	An integer, 0 through 65535.

*CLS (Clear Status)

Command

```
*CLS
```

The *CLS common command clears all event status registers, queues, and data structures, including the device defined error queue and status byte. If the *CLS command immediately follows a <program message terminator>, the output queue and the MAV (Message Available) bit will be cleared. Refer to chapter 6, "Status Reporting," for a complete discussion of status.

Example

```
OUTPUT XXX;"*CLS"
```

*ESE (Event Status Enable)

Command

```
*ESE <mask>
```

The *ESE command sets the Standard Event Status Enable Register bits. The Standard Event Status Enable Register contains a bit to enable the status indicators detailed in table 8-2. A 1 in any bit position of the Standard Event Status Enable Register enables the corresponding status in the Standard Event Status Enable Register. Refer to Chapter 6, "Status Reporting" for a complete discussion of status.

`<mask>` An integer from 0 to 255

Example

In this example, the *ESE 32 command will enable CME (Command Error), bit 5 of the Standard Event Status Enable Register. Therefore, when a command error occurs, the event summary bit (ESB) in the Status Byte Register will also be set.

```
OUTPUT XXX;"*ESE 32"
```

Query

```
*ESE?
```

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

Returned Format

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

Example

```
OUTPUT XXX;"*ESE?"
```

Table 8-2 **Standard Event Status Enable Register**

Bit Position	Bit Weight	Enables
7	128	PON - Power On
6	64	URQ - User Request
5	32	CME - Command Error
4	16	EXE - Execution Error
3	8	DDE - Device Dependent Error
2	4	QYE - Query Error
1	2	RQC - Request Control
0	1	OPC - Operation Complete

*ESR (Event Status Register)

Query

```
*ESR?
```

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

Returned Format

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

`<status>` An integer from 0 to 255

Example

If a command error has occurred, and bit 5 of the ESE register is set, the string variable Esr_event$ will have bit 5 (the CME bit) set.

```
10 OUTPUT XXX;"*ESE 32          !Enables bit 5 of the status register
20 OUTPUT XXX;"*ESR?"           !Queries the status register
30 ENTER XXX; Esr_event$        !Reads the query buffer
```

Table 8-3 shows the Standard Event Status Register. The table details the meaning of each bit position in the Standard Event Status Register and the bit weight. When you read Standard Event Status Register, the value returned is the total bit weight of all the bits that are high at the time you read the byte.

Table 8-3 The Standard Event Status Register

Bit Position	Bit Weight	Bit Name	Condition
7	128	PON	0 = register read - not in power up mode 1 = power up
6	64	URQ	0 = user request - not used - always zero
5	32	CME	0 = no command errors 1 = a command eror has been detected
4	16	EXE	0 = no execution errors 1 = an execution error has been detected
3	8	DDE	0 = no device dependent error has been detected 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 zero
0	1	OPC	0 = operation is not complete 1 = operation is complete

*IDN (Identification Number)

Query `*IDN?`

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

`"HEWLETT-PACKARD,1660A,0,REV <revision_code>"`

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

Returned Format `HEWLETT-PACKARD,1660A,0,REV <revision code>`

`<revision code>` Four digit-code in the format `XX.XX` representing the current ROM revision.

Example

```
OUTPUT XXX;"*IDN?"
```

*IST (Individual Status)

Query `*IST?`

The *IST query allows the instrument to identify itself during parallel poll by allowing the controller to read the current state of the IEEE 488.1 defined "ist" local message in the instrument. The response to this query is dependent upon the current status of the instrument.

Figure 8-2 shows the *IST data structure.

Returned Format `<id><NL>`

`<id>` 0 or 1

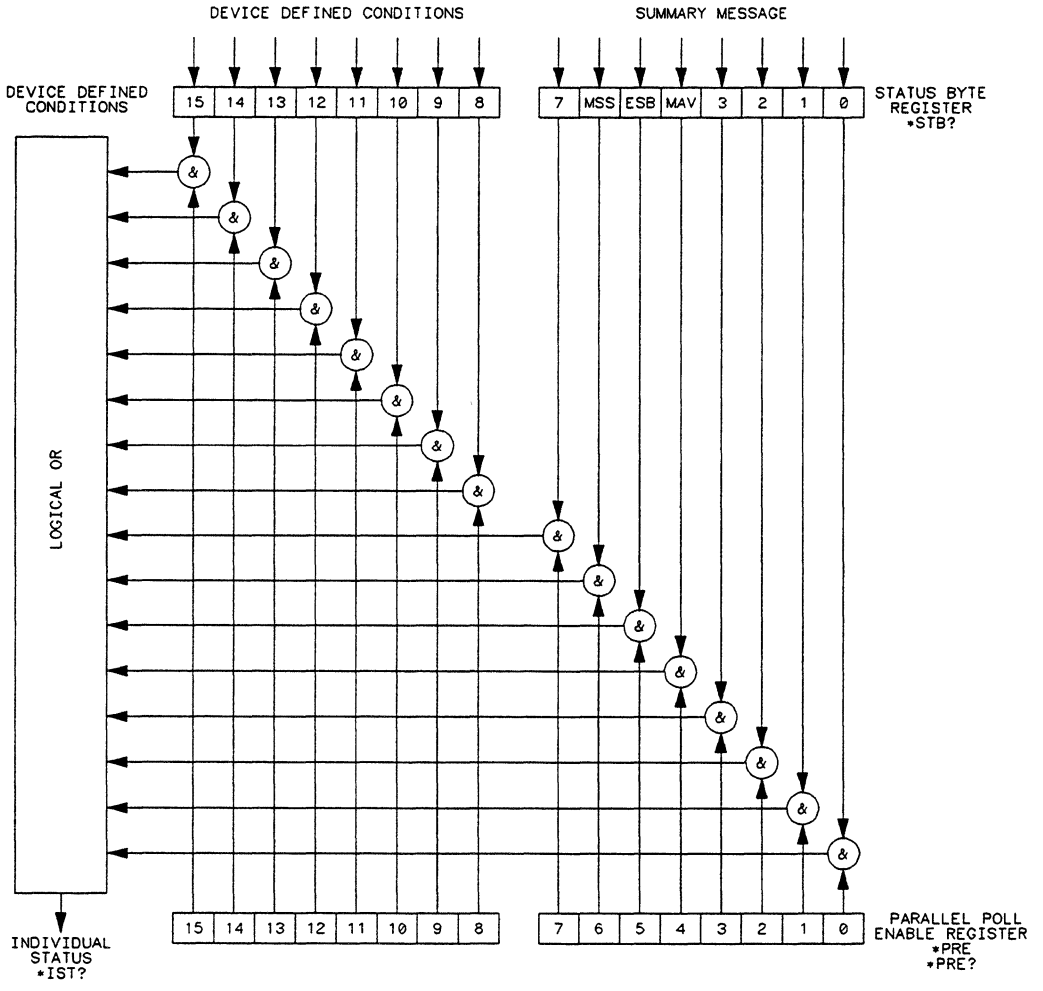
1 Indicates the "ist" local message is false.

0 Indicates the "ist" local message is true.

Example

```
OUTPUT XXX;"*IST?"
```

Figure 8-2



***IST Data Structure**

*OPC (Operation Complete)

Command

```
*OPC
```

The *OPC command will cause the instrument to set the operation complete bit in the Standard Event Status Register when all pending device operations have finished. The commands which affect this bit are the overlapped commands. An overlapped command is a command that allows execution of subsequent commands while the device operations initiated by the overlapped command are still in progress. The overlapped commands for the HP 1660-series are STARt and STOP.

Example

```
OUTPUT XXX;"*OPC"
```

Query

```
*OPC?
```

The *OPC query places an ASCII "1" in the output queue when all pending device operations have been completed.

Returned Format

```
1<NL>
```

Example

```
OUTPUT XXX;"*OPC?"
```

*OPT (Option Identification)

Query

```
*OPT?
```

The *OPT query identifies the software installed in the HP 1660-series. This query returns nine parameters. The first parameter indicates whether you are in the system. The next two parameters indicate any software options installed, and the next parameter indicates whether intermodule is available for the system. The last five parameters list the installed software for the modules in slot A through E for an HP 16500A mainframe. However, the HP 1660-series logic analyzers have only one slot (A); therefore, only the first parameter of the the last five parameters will be relevant. A zero in any of the last eight parameters indicates that the corresponding software is not currently installed. The name returned for software options and module software is the same name that appears in the field in the upper-left corner of the menu for each option or module.

Returned Format

```
{SYSTEM},{<option>|0},{<option>|0},{INTERMODULE|0},{<module>|0},
{<module>|0},{<module>|0},{<module>|0},{<module>|0}<NL>
```

`<option>` Name of software option.

`<module>` Name of module software.

Example

```
OUTPUT XXX;"*OPT?"
```

*PRE (Parallel Poll Enable Register Enable)

Command

```
*PRE <mask>
```

The *PRE command sets the parallel poll register enable bits. The Parallel Poll Enable Register contains a mask value that is ANDed with the bits in the Status Bit Register to enable an "ist" during a parallel poll. Refer to table 8-4 for the bits in the Parallel Poll Enable Register and for what they mask.

`<pre_mask>` An integer from 0 to 65535.

Example

This example will allow the HP 1660-series to generate an "ist" when a message is available in the output queue. When a message is available, the MAV (Message Available) bit in the Status Byte Register will be high.

```
Output XXX;"*PRE 16"
```

Query

```
*PRE?
```

The *PRE? query returns the current value of the register.

Returned format

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

`<mask>` An integer from 0 through 65535 representing the sum of all bits that are set. .

Example

```
OUTPUT XXX;"*PRE?"
```

Table 8-4 **HP 1660-series Parallel Poll Enable Register**

Bit Position	Bit Weight	Enables
15 -8		Not used
7	128	Not used
6	64	MSS - Master Summary Status
5	32	ESB - Event Status
4	16	MAV - Message Available
3	8	LCL - Local
2	4	Not used
1	2	Not used
0	1	MSB - Module Summary

*RST (Reset)

The *RST command is not implemented on the HP 1660-series. The HP 1660-series will accept this command, but the command has no affect on the logic analyzer.

The *RST command is generally used to place the logic analyzer in a predefined state. Because the HP 1660-series allows you to store predefined configuration files for individual modules, or for the entire system, resetting the logic analyzer can be accomplished by simply loading the appropriate configuration file. For more information, refer to chapter 11, "MMEMory Subsystem."

*SRE (Service Request Enable)

Command `*SRE <mask>`

The *SRE command sets the Service Request Enable Register bits. The Service Request Enable Register contains a mask value for the bits to be enabled in the Status Byte Register. A one in the Service Request Enable Register will enable the corresponding bit in the Status Byte Register. A zero will disable the bit. Refer to table 8-5 for the bits in the Service Request Enable Register and what they mask.

Refer to Chapter 6, "Status Reporting," for a complete discussion of status.

`<mask>` An integer from 0 to 255

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 (Message Available) bit will be high.

```
OUTPUT XXX;"*SRE 16"
```

Query `*SRE?`

The *SRE query returns the current value.

Returned Format `<mask><NL>`

`<mask>` An integer from 0 to 255 representing the sum of all bits that are set.

Example

```
OUTPUT XXX;"*SRE?"
```

Table 8-5 **HP 1660-series Service Request Enable Register**

Bit Position	Bit Weight	Enables
15-8		not used
7	128	not used
6	64	MSS - Master Summary Status (always 0)
5	32	ESB - Event Status
4	16	MAV - Message Available
3	8	LCL- Local
2	4	not used
1	2	not used
0	1	MSB - Module Summary

*STB (Status Byte)

Query

```
*STB?
```

The *STB query returns the current value of the instrument's status byte. The MSS (Master Summary Status) bit, and, not the RQS (Request Service) bit is reported on bit 6. The MSS indicates whether or not the device has at least one reason for requesting service. Refer to table 8-6 for the meaning of the bits in the status byte.

Refer to Chapter 6, "Status Reporting" for a complete discussion of status.

Returned Format

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

`<value>` An integer from 0 through 255

Example

```
OUTPUT XXX;"*STB?"
```

Table 8-6 **The Status Byte Register**

Bit Position	Bit Weight	Bit Name	Condition
7	128		0 = not Used
6	64	MSS	0 = instrument has no reason for service 1 = instrument is requesting service
5	32	ESB	0 = no event status conditions have occurred 1 = an enabled event status condition has occurred
4	16	MAV	0 = no output messages are ready 1 = an output message is ready
3	8	LCL	0 = a remote-to-local transition has not occurred 1 = a remote-to-local transition has occurred
2	4		not used
1	2		not used
0	1	MSB	0 = a module or the system has activity to report 1 = no activity to report

0 = False = Low
1 = True = High

*TRG (Trigger)

Command

```
*TRG
```

The *TRG command has the same effect as a Group Execute Trigger (GET). That effect is as if the START command had been sent for intermodule group run. If no modules are configured in the Intermodule menu, this command has no effect.

Example

```
OUTPUT XXX;"*TRG"
```

*TST (Test)

Query

```
*TST?
```

The *TST query returns the results of the power-up self-test. The result of that test is a 9-bit mapped value which is placed in the output queue. A one in the corresponding bit means that the test failed and a zero in the corresponding bit means that the test passed. Refer to table 8-7 for the meaning of the bits returned by a TST? query.

Returned Format

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

`<result>` An integer 0 through 511

Example

```
10 OUTPUT XXX;"*TST?"
20 ENTER XXX;Tst_value
```

Table 8-7 Bits Returned by *TST? Query (Power-Up Test Results)

Bit Position	Bit Weight	Test
8	256	Disk Test
7	128	not used
6	64	not used
5	32	Front-panel Test
4	16	HIL Test
3	8	Display Test
2	4	Interupt Test
1	2	RAM Test
0	1	ROM Test

*WAI (Wait)

Command

```
*WAI
```

The *WAI command causes the device to wait until completing all of the overlapped commands before executing any further commands or queries. An overlapped command is a command that allows execution of subsequent commands while the device operations initiated by the overlapped command are still in progress. Some examples of overlapped commands for the HP 1660-series are STARt and STOP.

Example:

```
OUTPUT XXX;"*WAI"
```

9

Mainframe Commands

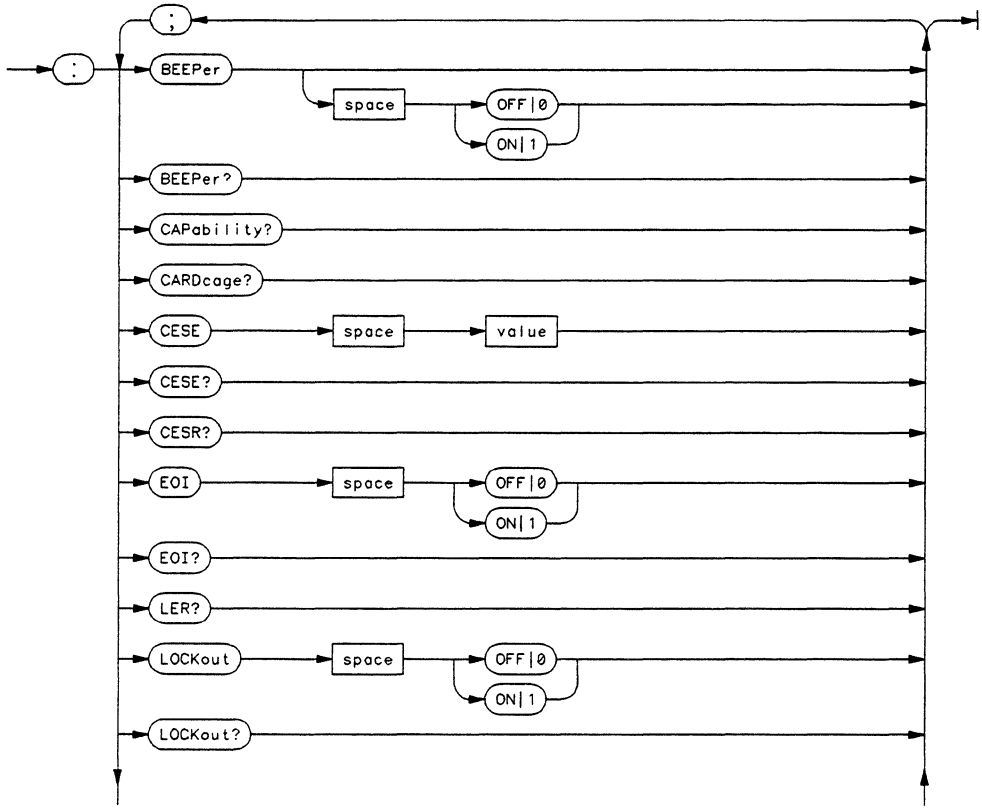
Introduction

Mainframe commands control the basic operation of the instrument for the HP 1660-series logic analyzers. The 1660-series logic analyzers are similar to an HP 16500A logic analysis system with a single logic analyzer module installed. The main difference in mainframe commands for the HP 1660-series logic analyzers is the number of modules. In the HP 1660 series, module 0 contains the system level commands and module 1 contains the logic analyzer level commands.

The command parser in the HP 1660-series logic analyzers is designed to accept programs written for the HP 16500A logic analysis system with an HP 16550A logic analyzer module. The main difference is how you specify the SELECT command. Remember, the HP 1660 series is equivalent only to a mainframe with one module; therefore, if you specify 2 through 10 for the SELECT command in your program, the command parser will take no action.

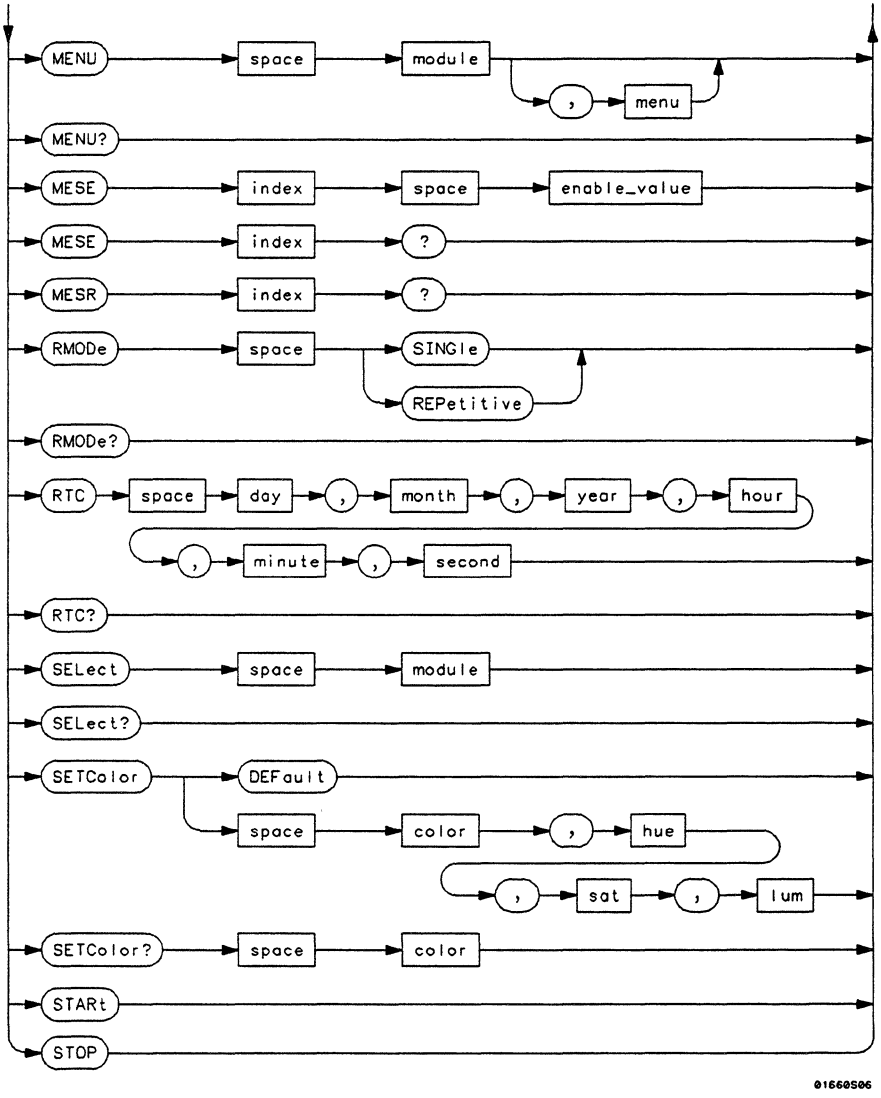
This chapter contains the mainframe commands with a syntax example for each command. Each syntax example contains the parameters for the HP 1600 series only. Refer to figure 9-1 and table 9-1 for the Mainframe commands syntax diagram.

Figure 9-1



Mainframe Commands Syntax Diagram

Figure 9-1 (continued)



Mainframe Commands Syntax Diagram (continued)

Table 9-1 **Mainframe Parameter Values**

Parameter	Values
value	An integer from 0 to 65535.
module	An integer 0 or 1 (2 through 10 unused).
menu	An integer.
enable_value	An integer from 0 to 255.
index	An integer from 0 to 5.
day	An integer from 1 through 31
month	An integer from 1 through 12
year	An integer from 1990 through 2089
hour	An integer from 0 through 23
minute	An integer from 0 through 59
second	An integer from 0 through 59
color	An integer from 1 to 7.
hue	An integer from 0 to 100 (always 0 for HP 1660 series).
sat	An integer from 0 to 100 (always 0 for HP 1660 series).
lum	An integer from 0 to 100.

BEEPer

Command

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

The BEEPer command sets the beeper mode, which turns the beeper sound of the instrument on and off. When BEEPer is sent with no argument, the beeper will be sounded without affecting the current mode.

Example

```
OUTPUT XXX;":BEEPER"
OUTPUT XXX;":BEEP ON"
```

Query

```
:BEEPer?
```

The BEEPer? query returns the mode currently selected.

Returned Format

```
[:BEEPer] {1|0}<NL>
```

Example

```
OUTPUT XXX;":BEEPER?"
```

CAPability

Query

```
:CAPability?
```

The CAPability query returns the HP-SL (HP System Language) and lower level capability sets implemented in the device.

Table 9-2 lists the capability sets implemented in the HP 1660-series.

Returned Format

```
[:CAPability] IEEE488,1987,SH1,AH1,T5,L4,SR1,RL1,PP1,DC1,
DT1,C0,E2<NL>
```

Example

```
OUTPUT XXX;":CAPABILITY?"
```

Table 9-2 HP 1660-series Capability Sets

Mnemonic	Capability Name	Implementation
SH	Source Handshake	SH1
AH	Acceptor Handshake	AH1
T	Talker (or TE - Extended Talker)	T5
L	Listener (or LE - Extended Listener)	L4
SR	Service Request	SR1
RL	Remote Local	RL1
PP	Parallel Poll	PP1
DC	Device Clear	DC1
DT	Device Trigger	DT1
C	Any Controller	C0
E	Electrical Characteristic	E2

CARDcage

Query

```
:CARDcage?
```

The CARDcage query returns a series of integers which identifies the modules that are installed in the mainframe. For an HP 1660-series, the first number returned is the card identification number and will always be 32. The second number returned indicates the module assignment for the logic analyzer and will always be 1. The possible values for the module assignment are 0 and 1 where 0 indicates the module software is not recognized or not loaded.

Returned Format

```
[:CARDcage] <ID>,<ID>,<ID>,<ID>,<ID><assign>,<assign>,
<assign>,<assign>,<assign><NL>
```

`<ID>` An integer indicating the card identification number.

`<assign>` An integer indicating the module assignment.

Example

```
OUTPUT XXX;":CARDCAGE?"
```

CESE (Combined Event Status Enable)

Command

```
:CESE <value>
```

The CESE command sets the Combined Event Status Enable register. This register is the enable register for the CESR register and contains the combined status of all of the MESE (Module Event Status Enable) registers of the HP 1660-series. Table 9-3 lists the bit values for the CESE register.

`<value>` An integer from 0 to 65535

Example

```
OUTPUT XXX;":CESE 32"
```

Query

```
:CESE?
```

The CESE? query returns the current setting.

Returned Format

```
[:CESE] <value><NL>
```

Example

```
OUTPUT XXX;":CESE?"
```

Table 9-3 HP 1660-series Combined Event Status Enable Register

Bit	Weight	Enables
2 to 15		not used
1	2	logic analyzer
0	1	Intermodule

CESR (Combined Event Status Register)

Query

```
:CESR?
```

The CESR query returns the contents of the Combined Event Status register. This register contains the combined status of all of the MESRs (Module Event Status Registers) of the HP 1660-series. Table 9-4 lists the bit values for the CESR register.

Returned Format

```
[:CESR] <value><NL>
```

`<value>` An integer from 0 to 65535

Example

```
OUTPUT XXX;":CESR?"
```

Table 9-4 HP 1660-series Combined Event Status Register

Bit	Bit Weight	Bit Name	Condition
2 to 15			0 = not used
1	2	Logic analyzer	0 = No new status 1 = Status to report
0	1	Intermodule	0 = No new status 1 = Status to report

EOI (End Or Identify)

Command

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

The EOI command specifies whether or not the last byte of a reply from the instrument is to be sent with the EOI bus control line set true or not. If EOI is turned off, the logic analyzer will no longer be sending IEEE 488.2 compliant responses.

Example

```
OUTPUT XXX;":EOI ON"
```

Query

```
:EOI?
```

The EOI? query returns the current status of EOI.

Returned Format

```
[:EOI] {1|0}<NL>
```

Example

```
OUTPUT XXX;":EOI?"
```

LER (LCL Event Register)

Query

```
:LER?
```

The LER query allows the LCL Event Register to be read. After the LCL Event Register is read, it is cleared. A one indicates a remote-to-local transition has taken place. A zero indicates a remote-to-local transition has not taken place.

Returned Format

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

Example

```
OUTPUT XXX;":LER?"
```

LOCKout

Command

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

The LOCKout command locks out or restores front panel operation. When this function is on, all controls (except the power switch) are entirely locked out.

Example

```
OUTPUT XXX;":LOCKOUT ON"
```

Query

```
:LOCKout?
```

The LOCKout query returns the current status of the LOCKout command.

Returned Format

```
[:LOCKout] {0|1}<NL>
```

Example

```
OUTPUT XXX;":LOCKOUT?"
```

MENU

Command

```
:MENU <module>[,<menu>]
```

The MENU command puts a menu on the display. The first parameter specifies the desired module. The optional second parameter specifies the desired menu in the module (defaults to 0). Table 9-5 lists the parameters and the menus.

`<module>` Selects module or system (integer) 0 selects the system, 1 selects the logic analyzer. –2, –1 and 2 to 10 unused)

`<menu>` Selects menu (integer)

Example

```
OUTPUT XXX;":MENU 0,1"
```

Table 9-5 Menu Parameter Values

Parameters	Menu
0,0	System RS-232/HP-IB
0,2	System Disk
0,3	System Utilities
0,4	System Test
1,0	Analyzer Configuration
1,1	Format 1
1,2	Format 2
1,3	Trigger 1
1,4	Trigger 2
1,5	Waveform 1
1,6	Waveform 2
1,7	Listing 1
1,8	Listing 2
1,9	Mixed
1,10	Compare 1
1,11	Compare 2
1,12	Chart 1
1,13	Chart 2

Query `:MENU?`

The MENU query returns the current menu selection.

Returned Format `[:MENU] <module>,<menu><NL>`

Example

```
OUTPUT XXX;":MENU?"
```

MESE<N> (Module Event Status Enable)

Command `:MESE<N> <enable_value>`

The MESE command sets the Module Event Status Enable register. This register is the enable register for the MESR register. The <N> index specifies the module, and the parameter specifies the enable value. For the HP 1660 series, the <N> index 0 and 1 refers to system and logic analyzer respectively.

`<N>` An integer 0 or 1 (2 through 10 unused).

`<enable_value>` An integer from 0 through 255

Example

```
OUTPUT XXX;":MESE1 3"
```

Query `:MESE<N>?`

The query returns the current setting. Tables 9-6 and 9-7 list the Module Event Status Enable register bits, bit weights, and what each bit masks for the mainframe and logic analyzer respectively.

Returned Format `[:MESE<N>] <enable_value><NL>`

Example

```
OUTPUT XXX;":MESE1?"
```

Table 9-6 **HP 1660-series Mainframe (Intermodule) Module Event Status Enable Register**

Bit Position	Bit Weight	Enables
7	128	not used
6	84	not used
5	32	not used
4	16	not used
3	8	not used
2	4	not used
1	2	RNT - Intermodule Run Until Satisfied
0	1	MC - Intermodule Measurement Complete

Table 9-7 **HP 1660-series Logic Analyzer Module Event Status Enable Register**

Bit Position	Bit Weight	Enables
7	128	not used
6	84	not used
5	32	not used
4	16	not used
3	8	Pattern searches failed
2	4	Trigger found
1	2	RNT - Run Until Satisfied
0	1	MC - Measurement Complete

MESR<N> (Module Event Status Register)

Query

```
:MESR<N>?
```

The MESR query returns the contents of the Module Event Status register. The <N> index specifies the module. For the HP 1660 series, the <N> index 0 and 1 refers to system and logic analyzer respectively.

Refer to table 9-8 for information about the Module Event Status Register bits and their bit weights for the system and table 9-9 for the logic analyzer.

Returned Format

```
[:MESR<N>] <enable_value><NL>
```

`<N>` An integer 0 through 10 (2 through 10 unused).

`<enable_value>` An integer from 0 through 255

Example

```
OUTPUT XXX;":MESR1?"
```

Table 9-8 **HP 1660-series Mainframe Module Event Status Register**

Bit	Bit Weight	Bit Name	Condition
7	128		0 = not used
6	64		0 = not used
5	32		0 = not used
4	16		0 = not used
3	8		0 = not used
2	4		0 = not used
1	2	RNT	0 = Intermodule Run until not satisfied 1 = Intermodule Run until satisfied
0	1	MC	0 = Intermodule Measurement not satisfied 1 = Intermodule Measurement satisfied

Table 9-9 **HP 1660-series Logic Analyzer Module Event Status Register**

Bit	Bit Weight	Condition
7	128	0 = not used
6	64	0 = not used
5	32	0 = not used
4	16	0 = not used
3	8	1 = One or more pattern searches failed 0 = Pattern searches did not fail
2	4	1 = Trigger found 0 = Trigger not found
1	2	0 = Run until not satisfied 1 = Run until satisfied
0	1	0 = Measurement not satisfied 1 = Measurement satisfied

RMODe

Command

```
:RMODe {SINGle|REPetitive}
```

The RMODe command specifies the run mode for the selected module (or Intermodule). If the selected module is in the intermodule configuration, then the intermodule run mode will be set by this command.

> After specifying the run mode, use the STARt command to start the acquisition.

Example

```
OUTPUT XXX;":RMODE SINGLE"
```

Query

```
:RMODe?
```

The query returns the current setting.

Returned Format

```
[:RMODe] {SINGle|REPetitive}<NL>
```

Example

```
OUTPUT XXX;":RMODE?"
```

RTC (Real-time Clock)

Command

```
:RTC {<day>,<month>,<year>,<hour>,<minute>,
<second>|DEFault}
```

The real-time clock command allows you to set the real-time clock to the current date and time. The DEFault option sets the real-time clock to 01 January 1990, 12:00:00 (24-hour format).

`<day>` integer from 1 to 31

`<month>` integer from 1 to 12

`<year>` integer from 1990 to 2089

`<hour>` integer from 0 to 23

`<minute>` integer from 0 to 59

`<second>` integer from 0 to 59

Example

This example sets the real-time clock for 1 January 1992, 20:00:00 (8 PM).

```
OUTPUT XXX;":RTC 1,1,1992,20,0,0"
```

Query `:RTC?`

The RTC query returns the real-time clock setting.

Returned Format `[:RTC] <day>,<month>,<year>,<hour>,<minute>,<second>`

Example

```
OUTPUT XXX;":RTC?"
```

SELect

Command `:SELect <module>`

The SELect command selects which module (or system) will have parser control. SELect defaults to System (0) at power up. The appropriate module (or system) must be selected before any module (or system) specific commands can be sent. SELECT 0 selects the System, SELECT 1 selects the logic analyzer (state and timing). Select –2, –1 and, 2 through 10 are accepted but no action will be taken. When a module is selected, the parser recognizes the module's commands and the System/Intermodule commands. When SELECT 0 is used, only the System/Intermodule commands are recognized by the parser. Figure 9-2 shows the command tree for the SELect command.

> The command parser in the HP 1660-series is designed to accept programs written for the HP 16500A logic analysis system with an HP 16550A logic analyzer module; however, if the parameters 2 through 10 are sent, the HP 1660-series will take no action.

`<module>` An integer 0 or 1 (–2, –1, and 2 through 10 unused).

Example

```
OUTPUT XXX;":SELECT 0"
```

Query `:SELect?`

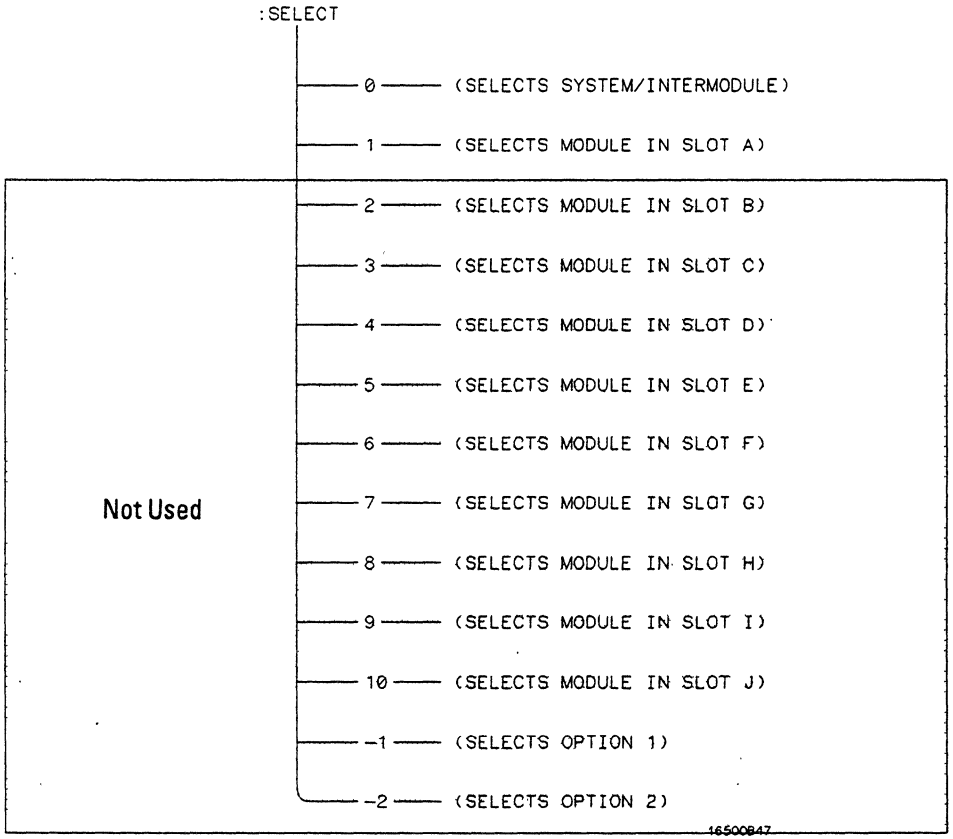
The SELect? query returns the current module selection.

Returned Format `[:SELect] <module><NL>`

Example

```
OUTPUT XXX;":SELECT?"
```

Figure 9-2



Select Command Tree

SETColor

Command

```
:SETColor {<color>,<hue>,<sat>,<lum>|DEFault}
```

The SETColor command is used to change one of the selections on the CRT, or to return to the default screen colors. Because the HP 1660-series display is monochrome, SETColor controls the grey-scale and brightness only. Four parameters are sent with the command to change a color:

- Color Number (first parameter)
- Hue (second parameter)
- Saturation (third parameter)
- Luminosity (last parameter)

> The command parser in the HP 1660-series is designed to accept programs written for the HP 16500A logic analysis system with an HP 16550A logic analyzer module. However, parameters for hue and saturation must be sent so that the parser recognizes the correct number of parameters; but, they are otherwise ignored by the HP 1660-series.

`<color>` An integer from 1 to 7

`<hue>` An integer from 0 to 100. Always 0 when returned by the query.

`<sat>` An integer from 0 to 100. Always 0 when returned by the query.

`<lum>` An integer from 0 to 100

> Color Number 0 cannot be changed.

Example

```
OUTPUT XXX;":SETCOLOR 3,0,0,60"
OUTPUT XXX;":SETC DEFAULT"
```

Query

```
:SETColor? <color>
```

The SETColor query returns the luminosity values for a specified grey scale.

Returned Format

```
[:SETColor] <color>,0,0,<lum><NL>
```

Example

```
OUTPUT XXX;":SETCOLOR? 3"
```

STARt

Command

```
:STARt
```

The STARt command starts the selected module (or Intermodule) running in the specified run mode (see RMODe). If the specified module is in the Intermodule configuration, then the Intermodule run will be started.

> The STARt command is an overlapped command. An overlapped command is a command that allows execution of subsequent commands while the device operations initiated by the overlapped command are still in progress.

Example

```
OUTPUT XXX;":START"
```

STOP

Command

```
:STOP
```

The STOP command stops the selected module (or Intermodule). If the specified module is in the Intermodule configuration, then the Intermodule run will be stopped.

> The STOP command is an overlapped command. An overlapped command is a command that allows execution of subsequent commands while the device operations initiated by the overlapped command are still in progress.

Example

```
OUTPUT XXX;":STOP"
```

10

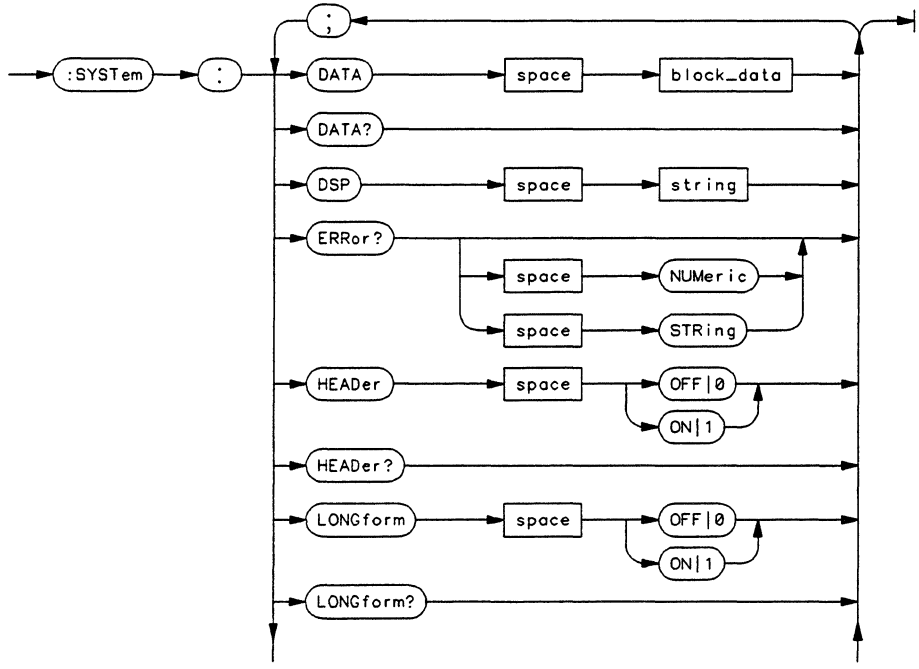
SYSTem Subsystem

Introduction

SYSTem subsystem commands control functions that are common to both analyzer 1 and analyzer 2, including formatting query responses and enabling reading and writing to the advisory line of the instrument. The command parser in the HP 1660-series is designed to accept programs written for the HP 16500A logic analysis system with an HP 16550A logic analyzer module.

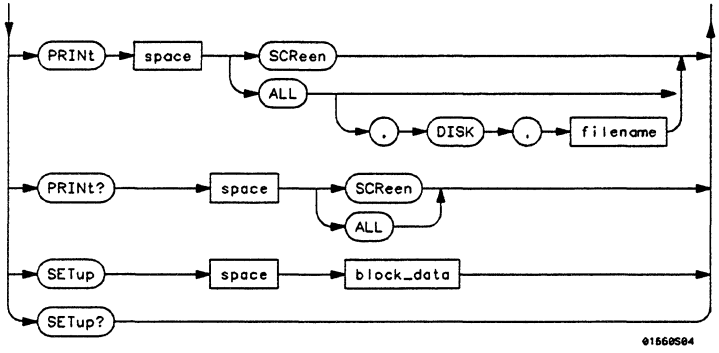
Refer to figure 10-1 and table 10-1 for the System Subsystem commands syntax diagram.

Figure 10-1



System Subsystem Commands Syntax Diagram

Figure 10-1



System Subsystem Commands Syntax Diagram (Continued)

Table 10-1 **SYSTem Parameter Values**

Parameter	Values
block_data	Data in IEEE 488.2 format.
string	A string of up to 68 alphanumeric characters.

DATA

Command

```
:SYSTem:DATA <block_data>
```

The DATA command allows you to send and receive acquired data to and from a controller in block form. This helps saving block data for:

- Reloading to the logic analyzer
- Processing data later in the logic analyzer
- Processing data in the controller.

The format and length of block data depends on the instruction being used and the configuration of the instrument. This chapter describes briefly the syntax of the Data command and query. Because of the capabilites and importance of the Data command and query, a complete chapter is dedicated to it. The dedicated chapter is chapter 26, "DATA and SETup Commands."

Example

```
OUTPUT XXX;":SYSTEM:DATA" <block_data>
```

`<block_data>`	`<block_length_specifier><section>`
`<block_length_specifier>`	`#8<length>`
`<length>`	The total length of all sections in byte format (must be represented with 8 digits)
`<section>`	`<section_header><section_data>`
`<section_header>`	16 bytes, described in the "Section Header Description" section in chapter 26.
`<section_data>`	The format depends on the type of data

Query

```
:SYSTem:DATA?
```

The SYSTem:DATA query returns the block data. The data sent by the SYSTem:DATA query reflects the configuration of the machines when the last run was performed. Any changes made since then through either front-panel operations or programming commands do not affect the stored configuration.

Returned Format

```
[:SYSTem:DATA] <block_data><NL>
```

Example

See "Transferring the logic analyzer acquired data" on page 27-17 in chapter 27, "Programming Examples" for an example.

DSP (Display)

Command

```
:SYSTem:DSP <string>
```

The DSP command writes the specified quoted string to a device-dependent portion of the instrument display.

`<string>` A string of up to 68 alphanumeric characters

Example

```
OUTPUT XXX;":SYSTEM:DSP 'The message goes here'"
```

ERRor

Query

```
:SYSTem:ERRor? [NUMeric|STRing]
```

The ERRor query returns the oldest error from the error queue. The optional parameter determines whether the error string should be returned along with the error number. If no parameter is received, or if the parameter is NUMeric, then only the error number is returned. If the value of the parameter is STRing, then the error should be returned in the following form:

```
<error_number>,<error_message (string)>
```

A complete list of error messages for the HP 1660A-series is shown in chapter 7, "Error Messages." If no errors are present in the error queue, a zero (No Error) is returned.

Returned Formats

Numeric:

```
[:SYSTem:ERRor] <error_number><NL>
```

String:

```
[:SYSTem:ERRor] <error_number>,<error_string><NL>
```

<error_number> An integer

<error_string> A string of alphanumeric characters

Examples

Numeric:

```
10 OUTPUT XXX;":SYSTEM:ERROR?"
20 ENTER XXX;Numeric
```

String:

```
50 OUTPUT XXX;":SYST:ERR? STRING"
60 ENTER XXX;String$
```

HEADer

Command

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

The HEADer command tells the instrument whether or not to output a header for query responses. When HEADer is set to ON, query responses will include the command header.

Example

```
OUTPUT XXX;":SYSTEM:HEADER ON"
```

Query

```
:SYSTem:HEADer?
```

The HEADer query returns the current state of the HEADer command.

Returned Format

```
[:SYSTem:HEADer] {1|0}<NL>
```

Example

```
OUTPUT XXX;":SYSTEM:HEADER?"
```

Headers should be turned off when returning values to numeric variables.

LONGform

Command

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

The LONGform command sets the longform variable, which tells the instrument how to format query responses. If the LONGform command is set to OFF, command headers and alpha arguments are sent from the instrument in the abbreviated form. If the the LONGform command is set to ON, the whole word will be output. This command has no affect on the input data messages to the instrument. Headers and arguments may be input in either the longform or shortform regardless of how the LONGform command is set.

Example

```
OUTPUT XXX;":SYSTEM:LONGFORM ON"
```

Query

```
:SYSTem:LONGform?
```

The query returns the status of the LONGform command.

Returned Format

```
[:SYSTem:LONGform] {1|0}<NL>
```

Example

```
OUTPUT XXX;":SYSTEM:LONGFORM?"
```

PRINt

Command

```
:SYSTem:PRINt {SCReen|ALL}[,DISK, <filename>]
```

The PRINt command initiates a print of the screen or listing buffer over the current PRINTER communication interface to the printer or to a file on the disk.

Example

This instuctrion prints the screen to the printer:

```
OUTPUT XXX;":SYSTEM:PRINT SCREEN"
```

This instruction prints all, for example the state listing, to a file with a filename STATE:

```
OUTPUT 707;":SYSTEM:PRINT ALL, DISK,'STATE'"
```

Query

```
:SYSTem:PRINt? {SCReen|ALL}
```

The PRINt query sends the screen or listing buffer data over the current CONTROLLER communication interface to the controller.

The print query should NOT be sent in conjunction with any other command or query on the same command line. The print query never returns a header. Also, since response data from a print query may be sent directly to a printer without modification, the data is not returned in block mode.

> PRINT? ALL is only available in menus that have the "Print All" option available on the front panel. For more information, refer to the *HP 1660-series User's Guide*.

Example

```
OUTPUT 707;":SYSTEM:PRINT? SCREEN"
```

SETup

Command

```
:SYStem:SETup <block_data>
```

The :SYStem:SETup command configures the logic analyzer module as defined by the block data sent by the controller. This chapter describes briefly the syntax of the Setup command and query. Because of the capabilites and importance of the Setup command and query, a complete chapter is dedicated to it. The dedicated chapter is chapter 26, "DATA and SETup Commands."

`<block_data>`	`<block_length_specifier><section>`
`<block_length_specifier>`	`#8<length>`
`<length>`	The total length of all sections in byte format (must be represented with 8 digits)
`<section>`	`<section_header><section_data>`
`<section_header>`	16 bytes, described in the "Section Header Description" section in chapter 26.
`<section_data>`	Format depends on the type of data

The total length of a section is 16 (for the section header) plus the length of the section data. So when calculating the value for `<length>`, don't forget to include the length of the section headers.

Example

```
OUTPUT XXX USING "#,K";":SYSTEM:SETUP  " <block_data>
```

Query

```
:SYStem:SETup?
```

The SYStem:SETup query returns a block of data that contains the current configuration to the controller.

Returned Format

```
[:SYStem:SETup] <block_data><NL>
```

Example

See "Transferring the logic analyzer configuration" on page 27-14 in chapter 27, "Programming Examples" for an example.

11

MMEMory Subsystem

Introduction

The MMEMory (mass memory) subsystem commands provide access to disk drive. The HP 1600-series logic analyzers support both LIF (Logical Information Format) and DOS (Disk Operating System) formats.

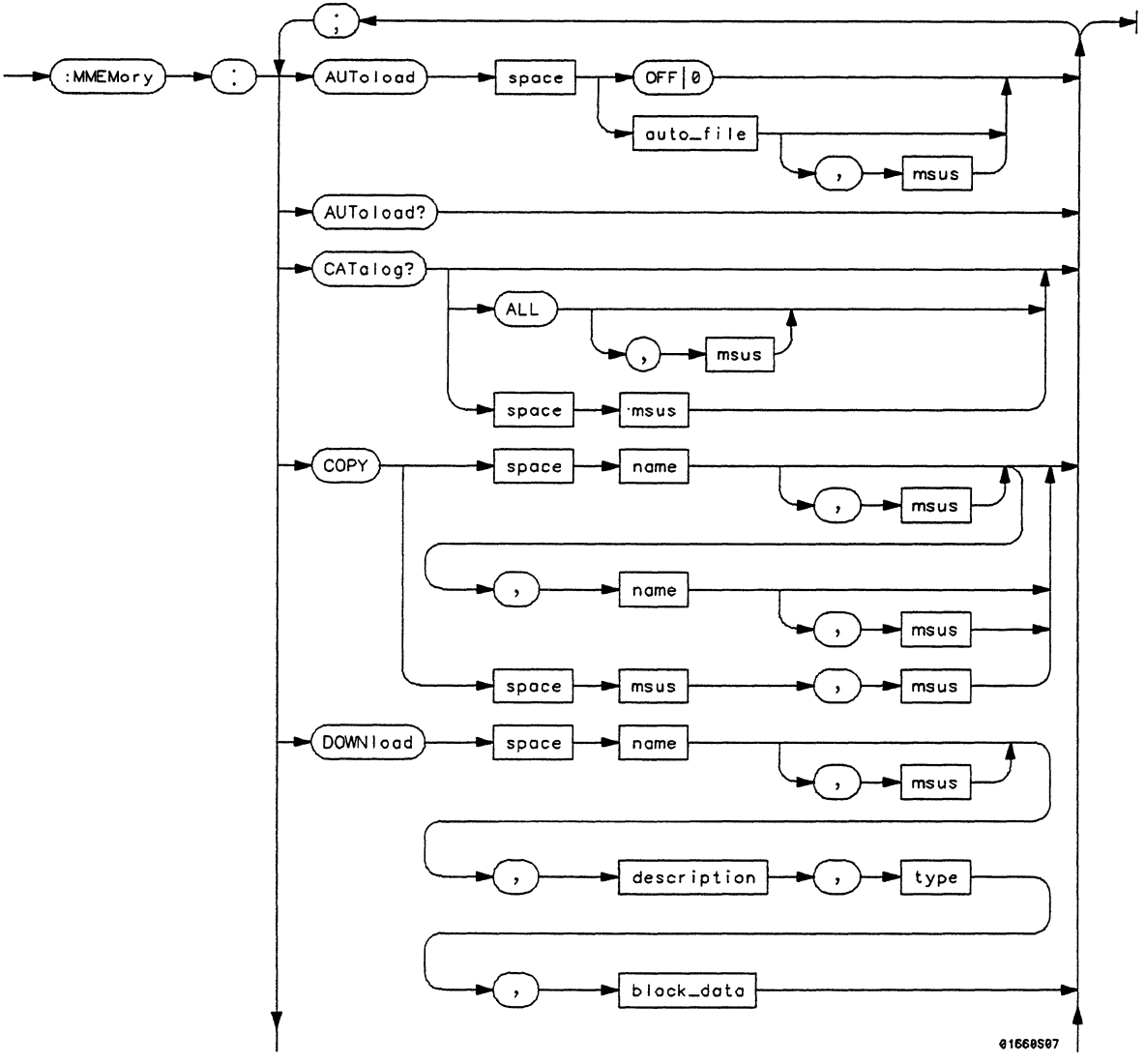
The HP 1660-series logic analyzers have only one disk drive; however, programs written for the HP 16500A logic analysis system that contain the MSI (Mass Storage Is) parameter will be accepted but no action is taken. Refer to figure 11-1 and table 11-1 for the MMEMory Subsystem commands syntax diagram. The MMEMory subsystem commands are:

- AUTOload
- CATalog
- COPY
- DOWNload
- INITialize
- LOAD
- MSI
- PACK
- PURGe
- REName
- STORe
- UPLoad
- VOLume

<msus> refers to the mass storage unit specifier; however, it is not needed for the HP 1660-series logic analyzers since they have only one drive. The <msus> parameter is shown in the command syntax examples as a reminder that for the the HP 16500A logic analysis system can be used on the HP 1660-series logic analyzers.

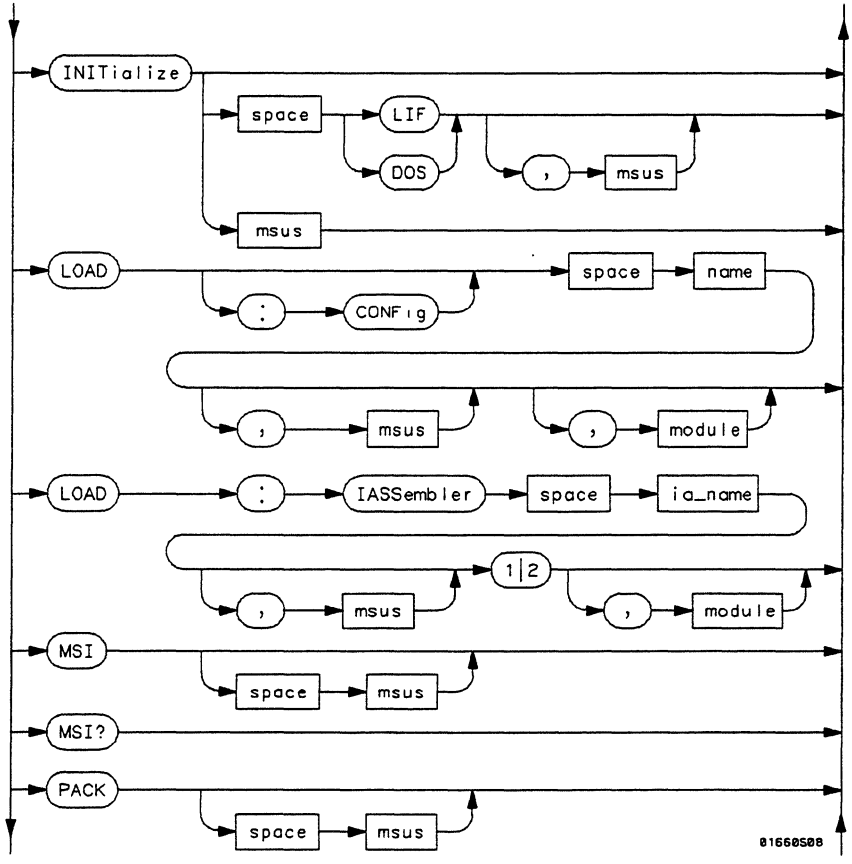
If you are not going to store information to the configuration disk, or if the disk you are using contains information you need, it is advisable to write protect your disk. This will protect the contents of the disk from accidental damage due to incorrect commands being mistakenly sent.

Figure 11-1



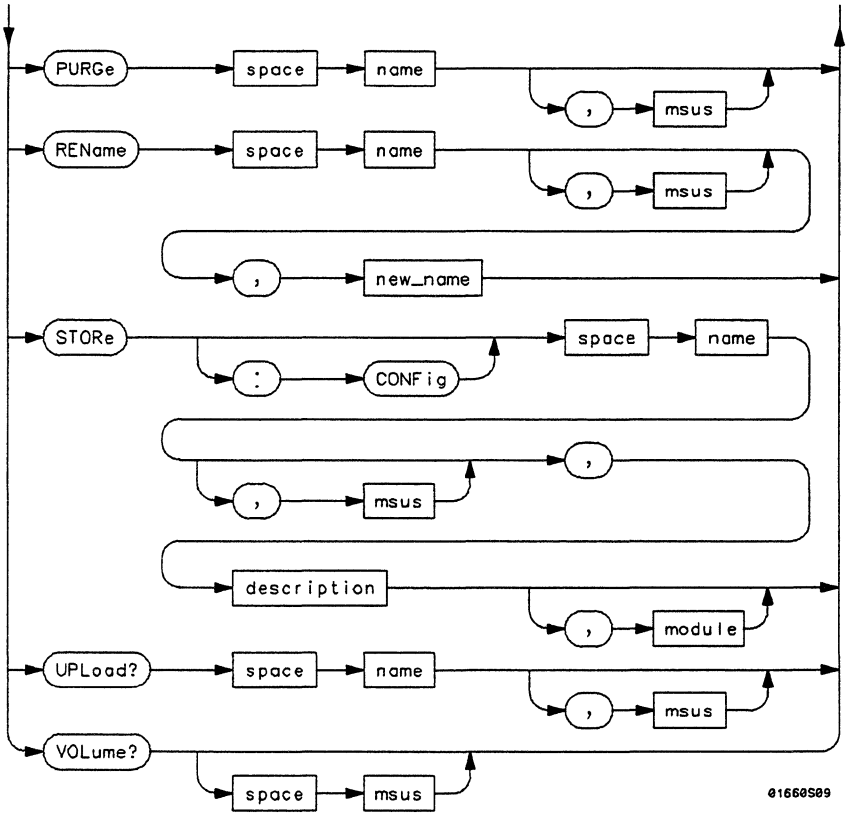
Mmemory Subsystem Commands Syntax Diagram

Figure 11-1



Mmemory Subsystem Commands Syntax Diagram (Continued)

Figure 11-1



Mmemory Subsystem Commands Syntax Diagram (Continued)

Table 11-1 MMEMory Parameter Values

Parameter	Values
auto_file	A string of up to 10 alphanumeric characters for LIF in the following form: "NNNNNNNNNN" or A string of up to 12 alphanumeric characters for DOS in the following form: "NNNNNNNN.NNN"
msus	Mass Storage Unit specifier (not needed by HP 1660-series. HP 16500A <msus> is accepted but no action is taken).
name	A string of up to 10 alphanumeric characters for LIF in the following form: "NNNNNNNNNN" or A string of up to 12 alphanumeric characters for DOS in the following form: "NNNNNNNN.NNN"
description	A string of up to 32 alphanumeric characters.
type	An integer, refer to table 11-2.
block_data	Data in IEEE 488.2 format.
ia_name	A string of up to 10 alphanumeric characters for LIF in the following form: "NNNNNNNNNN" or A string of up to 12 alphanumeric characters for DOS in the following form: "NNNNNNNN.NNN"
new_name	A string of up to 10 alphanumeric characters for LIF in the following form: "NNNNNNNNNN" or A string of up to 12 alphanumeric characters for DOS in the following form: "NNNNNNNN.NNN"
module	An integer, 0 or 1.

AUToload

Command

```
:MMEMory:AUToload {{OFF|0}|{<auto_file>}}[,<msus>]
```

The AUToload command controls the autoload feature which designates a set of configuration files to be loaded automatically the next time the instrument is turned on. The OFF parameter (or 0) disables the autoload feature. A string parameter may be specified instead to represent the desired autoload file. If the file is on the current disk, the autoload feature is enabled to the specified file.

`<auto_file>` A string of up to 10 alphanumeric characters for LIF in the following form:

`NNNNNNNNNN`

or

A string of up to 12 alphanumeric characters for DOS in the following form:

`NNNNNNNN.NNN`

`<msus>` Mass Storage Unit Specifier (not needed by HP 1660-series. HP 16500A <msus> is accepted but no action is taken).

Examples

```
OUTPUT XXX;":MMEMORY:AUTOLOAD OFF"
OUTPUT XXX;":MMEMORY:AUTOLOAD 'FILE1_A'"
OUTPUT XXX;":MMEMORY:AUTOLOAD 'FILE2 ',INTERNAL0"
```

Query

```
:MMEMory:AUToload?
```

The AUToload query returns 0 if the autoload feature is disabled. If the autoload feature is enabled, the query returns a string parameter that specifies the current autoload file. The appropriate slot designator is included in the filename and refers to the slot designator A for the logic analyzser. If the slot designator is _ (underscore) the file is for the system.

Returned Format

```
[:MMEMory:AUToload] {0|<auto_file>},<msus><NL>
```

<auto_file> A string of up to 10 alphanumeric characters for LIF in the following form:

NNNNNNNNNN

or

A string of up to 12 alphanumeric characters for DOS in the following form:

NNNNNNNN.NNN

Example

```
OUTPUT XXX;":MMEMORY:AUTOLOAD?"
```

CATalog

Query

```
:MMEMory:CATalog? [[All,][<msus>]]
```

The CATalog query returns the directory of the disk in one of two block data formats. The directory consists of a 51 character string for each file on the disk when the **ALL** option is not used. Each file entry is formatted as follows:

```
"NNNNNNNNNN TTTTTTT FFFFFFFFFFFFFFFFFFFFFFFFFFFFFF"
```

where N is the filename, T is the file type (see table 11-2), and F is the file description.

The optional parameter **ALL** returns the directory of the disk in a 70-character string as follows:

```
"NNNNNNNNNNNN TTTTTTT FFFFFFFFFFFFFFFFFFFFFFFFFFFFFF DDMMMYY
HH:MM:SS"
```

where N is the filename, T is the file type (see table 11-2), F is the file description, and, D, M, Y, and HH:MM:SS are the date, month, year, and time respectively in 24-hour format.

The **<msus>** is not needed by HP 1660-series; however, the HP 16500A <msus> is accepted but no action is taken.

`<msus>` Mass Storage Unit Specifier (not needed by HP 1660-series. HP 16500A <msus> is accepted but no action is taken).

Returned Format `[:MMEMory:CATalog] <block_data>`

`<block_data>` ASCII block containing `<filename> <file_type> <file_description>`

Example 1 This example is for sending the **CATALOG? ALL** query:

```
OUTPUT 707;":MMEMORY:CATALOG? ALL"
```

Example 2 This example is for sending the **CATALOG?** query without the ALL option. Keep in mind if you do not use the **ALL** option with a DOS disk, each filename entry will be truncated at 51 characters:

```
OUTPUT 707;":MMEMORY:CATALOG?"
```

COPY

Command `:MMEMory:COPY <name>[,<msus>],<new_name>[,<msus>]`

The COPY command copies one file to a new file or an entire disk's contents to another disk. The two **<name>** parameters are the filenames. The first pair of parameters specifies the source file. The second pair specifies the destination file. An error is generated if the source file doesn't exist, or if the destination file already exists.

The **<msus>** is not needed by HP 1660-series. HP 16500A <msus> is accepted but no action is taken.

`<name>` A string of up to 10 alphanumeric characters for LIF in the following form:

`NNNNNNNNNN`
or
A string of up to 12 alphanumeric characters for DOS in the following form:

`NNNNNNNN.NNN`

`<new_name>` A string of up to 10 alphanumeric characters for LIF in the following form:

`NNNNNNNNNN`
or
A string of up to 12 alphanumeric characters for DOS in the following form:

`NNNNNNNN.NNN`

`<msus>` Mass Storage Unit Specifier (not needed by HP 1660-series. HP 16500A `<msus>` is accepted but no action is taken).

Examples

To copy the contents of "FILE1" to "FILE2:

```
OUTPUT XXX;":MMEMORY:COPY 'FILE1','FILE2'"
```

DOWNload

Command

```
:MMEMory:DOWNload <name>[,<msus>],<description>,
<type>,<block_data>
```

The DOWNload command downloads a file to the mass storage device. The `<name>` parameter specifies the filename, the `<description>` parameter specifies the file descriptor, and the `<block_data>` contains the contents of the file to be downloaded.

The `<msus>` is not needed by HP 1660-series. HP 16500A `<msus>` is accepted but no action is taken.

Table 11-2 lists the file types for the `<type>` parameter.

`<name>` A string of up to 10 alphanumeric characters for LIF in the following form:

`NNNNNNNNNN`

or

A string of up to 12 alphanumeric characters for DOS in the following form:

`NNNNNNNN.NNN`

`<msus>` Mass Storage Unit Specifier (not needed by HP 1660-series. HP 16500A `<msus>` is accepted but no action is taken).

`<description>` A string of up to 32 alphanumeric characters

`<type>` An integer (see table 11-2)

`<block_data>` Contents of file in block data format

Example

```
OUTPUT XXX;":MMEMORY:DOWNLOAD 'SETUP  ',INTERNAL0,'FILE CREATED FROM SETUP
QUERY',-16127,#800000643..."
```

Table 11-2 File Types

File	File Type
HP 1660-series System Software	–15608
HP 1660-series ROM Software	–15609
HP 1660-series System Configuration	–15605
HP 1660-series Logic Analyzer Configuration	–16095
HP 1660-series Logic Analyzer Software	–15607
Autoload File	–15615
Inverse Assembler	–15614
Text Type (LIF from Print to Disk)	–5813

INITialize

Command

```
:MMEMory:INITialize [{LIF|DOS}[,<msus>]]
```

The INITialize command formats the disk in either LIF (Logical Information Format) or DOS (Disk Operating System). The `<msus>` is not needed by HP 1660-series. HP 16500A `<msus>` is accepted but no action is taken. If no format is specified, then the initialize command will format the disk in the LIF format.

`<msus>` Mass Storage Unit Specifier (not needed by HP 1660-series. HP 16500A `<msus>` is accepted but no action is taken).

Examples

```
OUTPUT XXX;":MMEMORY:INITIALIZE DOS"
OUTPUT XXX;":MMEMORY:INITIALIZE LIF,INTERNAL0"
```

Once executed, the initialize command formats the specified disk, permanently erasing all existing information from the disk. After that, there is no way to retrieve the original information.

LOAD [:CONFig]

Command

```
:MMEMory:LOAD[:CONfig] <name>[,<msus>][,<module>]
```

The LOAD command loads a configuration file from the disk into the logic analyzer, software options, or the system. The **<name>** parameter specifies the filename from the disk. The optional **<module>** parameter specifies which module(s) to load the file into. The accepted values are 0 for system and 1 for logic analyzer. Not specifying the **<module>** parameter is equivalent to performing a 'LOAD ALL' from the front panel which loads the appropriate file for both the system and logic analyzer, and any software option.

<name> A string of up to 10 alphanumeric characters for LIF in the following form:

NNNNNNNNNN

or

A string of up to 12 alphanumeric characters for DOS in the following form:

NNNNNNNN.NNN

<msus> Mass Storage Unit Specifier (not needed by HP 1660-series. HP 16500A **<msus>** is accepted but no action is taken).

<module> An integer, 0 or 1

Examples

```
OUTPUT XXX;":MMEMORY:LOAD:CONFIG 'FILE  '"
OUTPUT XXX;":MMEMORY:LOAD 'FILE  ',0"
OUTPUT XXX;":MMEM:LOAD:CONFIG 'FILE A',INTERNAL0,1"
```

LOAD :IASSembler

Command

```
:MMEMory:LOAD:IASSembler <IA_name>[,<msus>],{1|2}
[,<module>]
```

This variation of the LOAD command allows inverse assembler files to be loaded into a module that performs state analysis. The **<IA_name>** parameter specifies the inverse assembler filename from the desired **<msus>**. The parameter after the optional <msus> specifies which machine to load the inverse assembler into.

The optional **<module>** parameter is used to specify which slot the state analyzer in. 1 refers to the logic analyzer. If this parameter is not specified, the state analyzer will be selected.

<IA_name> A string of up to 10 alphanumeric characters for LIF in the following form:

NNNNNNNNNN

or

A string of up to 12 alphanumeric characters for DOS in the following form:

NNNNNNNN.NNN

<msus> Mass Storage Unit Specifier (not needed by HP 1660-series. HP 16500A **<msus>** is accepted but no action is taken).

<module> An integer, always 1

Examples

```
OUTPUT XXX;":MMEMORY:LOAD:IASSEMBLER 'I68020 IP',1"
OUTPUT XXX;":MMEM:LOAD:IASS 'I68020 IP',INTERNAL0,1,2"
```

MSI (Mass Storage Is)

Command

```
:MMEMory:MSI [<msus>]
```

The MSI command selects a default mass storage device; however it is not needed by HP 1660-series because it has only one disk drive. If the HP 16500A `<msus>` is sent to the HP 1660-series logic analyzer, it is accepted but no action is taken.

`<msus>` Mass Storage Unit Specifier (not needed by HP 1660-series. HP 16500A `<msus>` is accepted but no action is taken).

Examples

```
OUTPUT XXX;":MMEMORY:MSI"
OUTPUT XXX;":MMEM:MSI INTERNAL0"
```

Query

```
:MMEMory:MSI?
```

The MSI? query returns the current MSI setting. Because the HP 1660-series logic analyzers have only one disk drive, `Internal0` is always returned.

Returned Format

```
[:MMEMory:MSI] <msus><NL>
```

Example

```
OUTPUT XXX;":MMEMORY:MSI?"
```

PACK

Command

```
:MMEMory:PACK [<msus>]
```

The PACK command packs the files on the LIF disk the disk in the drive. If a DOS disk is in the drive when the PACK command is sent, no action is taken.

`<msus>` Mass Storage Unit Specifier (not needed by HP 1660-series. HP 16500A `<msus>` is accepted but no action is taken).

Examples

```
OUTPUT XXX;":MMEMORY:PACK"
OUTPUT XXX;":MMEM:PACK INTERNAL0"
```

PURGe

Command

```
:MMEMory:PURGe <name>[,<msus>]
```

The PURGe command deletes a file from the disk in the drive. The `<name>` parameter specifies the filename to be deleted.

`<name>` A string of up to 10 alphanumeric characters for LIF in the following form:

`NNNNNNNNNN`

or

A string of up to 12 alphanumeric characters for DOS in the following form:

`NNNNNNNN.NNN`

`<msus>` Mass Storage Unit Specifier (not needed by HP 1660-series. HP 16500A `<msus>` is accepted but no action is taken).

Examples

```
OUTPUT XXX;":MMEMORY:PURGE 'FILE1'"
OUTPUT XXX;":MMEM:PURG 'FILE1',INTERNAL0"
```

Once executed, the purge command permanently erases all the existing information about the specified file. After that, there is no way to retrieve the original information.

REName

Command

```
:MMEMory:REName <name>[,<msus>],<new_name>
```

The REName command renames a file on the disk in the drive. The **<name>** parameter specifies the filename to be changed and the **<new_name>** parameter specifies the new filename.

You cannot rename a file to an already existing filename.

<name> A string of up to 10 alphanumeric characters for LIF in the following form:

NNNNNNNNNN

or

A string of up to 12 alphanumeric characters for DOS in the following form:

NNNNNNNN.NNN

<msus> Mass Storage Unit Specifier (not needed by HP 1660-series. HP 16500A **<msus>** is accepted but no action is taken).

<new name> A string of up to 10 alphanumeric characters for LIF in the following form:

NNNNNNNNNN

or

A string of up to 12 alphanumeric characters for DOS in the following form:

NNNNNNNN.NNN

Examples

```
OUTPUT XXX;":MMEMORY:RENAME 'OLDFILE','NEWFILE'"
OUTPUT XXX;":MMEM:REN 'OLDFILE'[,INTERNAL1],'NEWFILE'"
```

STORe [:CONFig]

Command

```
:MMEMory:STORe [:CONfig]<name>[,<msus>],
<description>[,<module>]
```

The STORe command stores module or system configurations onto a disk. The [:CONFig] specifier is optional and has no effect on the command. The `<name>` parameter specifies the file on the disk. The `<description>` parameter describes the contents of the file. The optional `<module>` parameter allows you to store the configuration for either the system or the logic analyzer. 1 refers to the logic analyzer and 0 refers to the system.

If the optional `<module>` parameter is not specified, the configurations for both the system and logic analyzer are stored.

`<name>` A string of up to 10 alphanumeric characters for LIF in the following form:

`NNNNNNNNNN`

or

A string of up to 12 alphanumeric characters for DOS in the following form:

`NNNNNNNN.NNN`

`<msus>` Mass Storage Unit Specifier (not needed by HP 1660-series. HP 16500A `<msus>` is accepted but no action is taken).

`<description>` A string of up to 32 alphanumeric characters

`<module>` An integer, 0 or 1

Examples

```
OUTPUT XXX;":MMEM:STOR 'DEFAULTS','SETUPS FOR ALL MODULES'"
OUTPUT XXX;":MMEMORY:STORE:CONFIG  'STATEDATA',INTERNAL0,
'ANALYZER 1 CONFIG',1"
```

The appropriate module designator "_X" is added to all files when they are stored. "X" refers to either an __ (double underscore) for the system or an _A for the logic analyzer.

UPLoad

Query

```
:MMEMory:UPLoad? <name>[,<msus>]
```

The UPLoad query uploads a file. The <name> parameter specifies the file to be uploaded from the disk. The contents of the file are sent out of the instrument in block data form.

This command should only be used for HP 16550A or HP 1660-series configuration files.

<name> A string of up to 10 alphanumeric characters for LIF in the following form:

NNNNNNNNNN

or

A string of up to 12 alphanumeric characters for DOS in the following form:

NNNNNNNN.NNN

<msus> Mass Storage Unit Specifier (not needed by HP 1660-series. HP 16500A <msus> is accepted but no action is taken).

Returned Format

```
[:MMEMory:UPLoad] <block_data><NL>
```

Example

```
10 DIM Block$[32000]                          !allocate enough memory for block data
20 DIM Specifier$[2]
30 OUTPUT XXX;":EOI ON"
40 OUTPUT XXX;":SYSTEM HEAD OFF"
50 OUTPUT XXX;":MMEMORY:UPLOAD? 'FILE1'"      !send upload query
60 ENTER XXX USING "#,2A";Specifier$          !read in #8
70 ENTER XXX USING "#,8D";Length              !read in block length
80 ENTER XXX USING "-K";Block$                !read in file
90 END
```

VOLume

Query `:MMEMory:VOLume? [<msus>]`

TheVOLume query returns the volume type of the disk. The volume types are DOS or LIF. Question marks (???) are returned if there is no disk, if the disk is not formatted, or if a disk has a format other than DOS or LIF.

`<msus>` Mass Storage Unit Specifier (not needed by HP 1660-series. HP 16500A `<msus>` is accepted but no action is taken).

Returned Format `[:MMEMory:VOLume]{DOS|LIF|???}<NL>`

Example

```
OUTPUT XXX;":MMEMORY:VOLUME?"
```

12

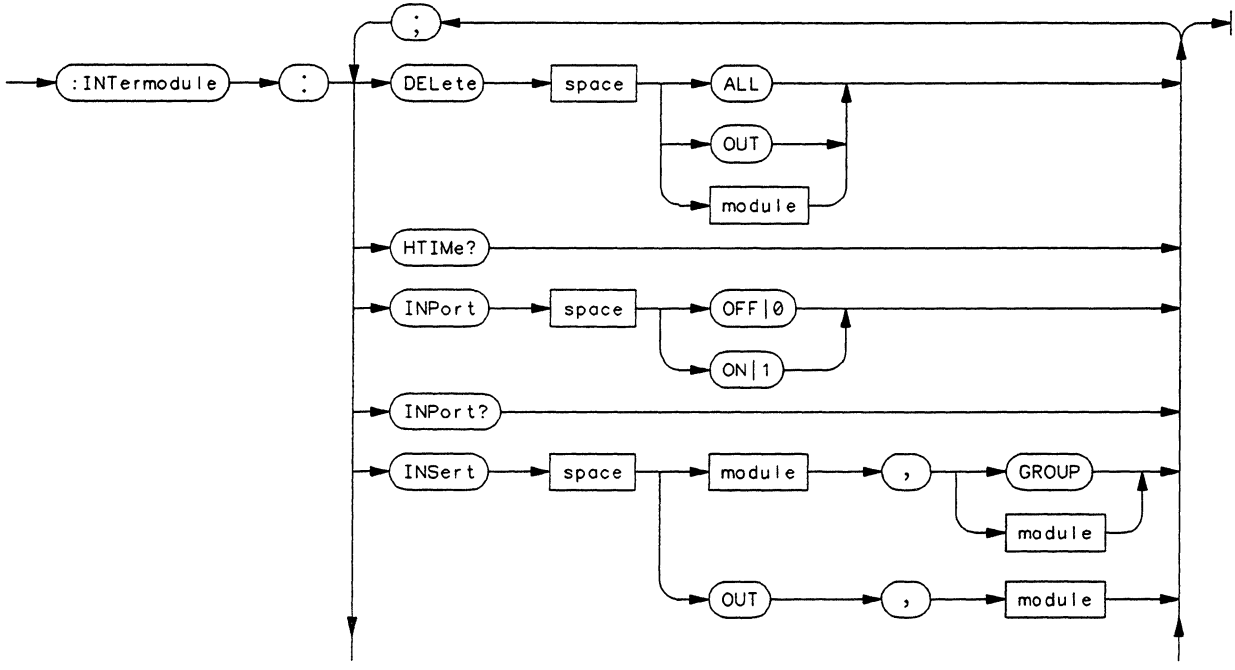
INTermodule Subsystem

Introduction

The INTermodule subsystem commands specify intermodule arming from the rear-panel input BNC (ARMIN) or to the rear-panel output BNC (ARMOUT). Refer to figure 12-1 and table 12-1 for the INTermodule Subsystem commands syntax diagram. The INTermodule commands are:

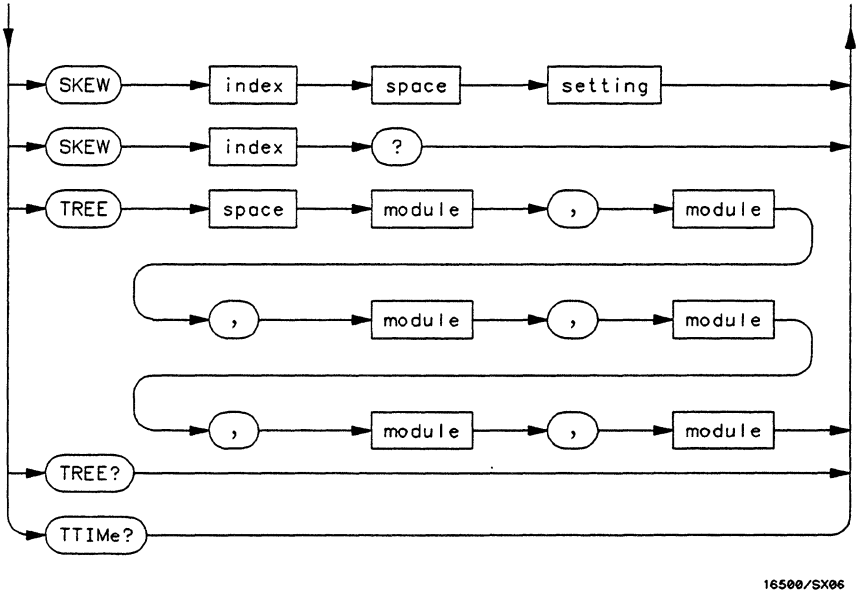
- DELete
- HTIMe
- INPort
- INSert
- SKEW
- TREE
- TTIMe

Figure 12-1



Intermodule Subsystem Commands Syntax Diagram

Figure 12-1



Intermodule Subsystem Commands Syntax Diagram (Continued)

Table 12-1 **INTermodule Parameter Values**

Parameter	Value
module	An integer, 1 to 10 (2 through 10 unused)
index	An integer, 1 to 10 (2 through 10 unused)
setting	A numeric, – 1.0 to 1.0 in seconds.

:INTermodule

Selector

```
:INTermodule
```

The INTermodule selector specifies INTermodule as the subsystem the commands or queries following will refer to. Because the INTermodule command is a root level command, it will normally appear as the first element of a compound header.

Example

```
OUTPUT XXX;":INTERMODULE:HTIME?"
```

DELete

Command

```
:DELete {ALL|OUT|<module>}
```

The DELete command is used to delete a module, PORT OUT, or an entire intermodule tree. The <module> parameter sent with the delete command refers to the slot location of the logic analyzer which is slot 1.

`<module>` An integer, 1 through 10 (2 through 10 unused)

Example

```
OUTPUT XXX;":INTERMODULE:DELETE ALL"
OUTPUT XXX;":INTERMODULE:DELETE 1"
```

HTIMe

Query

```
:HTIMe?
```

The HTIMe query returns a value representing the internal hardware skew in the Intermodule configuration. If there is no internal skew, 9.9E37 is returned.

> The internal hardware skew is only a display adjustment for time-correlated waveforms. The value returned is the average propagation delay of the trigger lines through the intermodule bus circuitry. The value is for reference only because the value returned by TTIMe includes the internal hardware skew represented by HTIMe.

Returned Format

```
[:INTermodule:HTIMe] <value><NL>
```

`<value>` Skew for logic analyzer (real number)

Example

```
OUTPUT XXX;":INTERMODULE:HTIME?"
```

INPort

Command

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

The INPort command causes intermodule acquisitions to be armed from the Input port.

Example

```
OUTPUT XXX;":INTERMODULE:INPORT ON"
```

Query `:INPort?`

The INPort query returns the current setting.

Returned Format `[:INTermodule:INPort] {1|0}<NL>`

Example

```
OUTPUT XXX;":INTERMODULE:INPORT?"
```

INSert

Command `:INSert {<module>|OUT},{GROUP|<module>}`

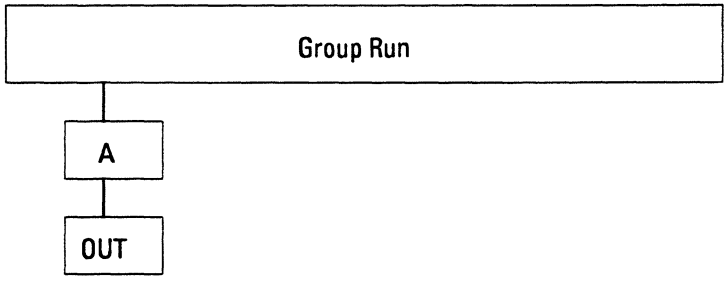
The INSert command adds PORT OUT to the Intermodule configuration. The first parameter selects the logic analyzer or PORT OUT to be added to the intermodule configuration, and the second parameter tells the instrument where the logic analyzer or PORT OUT will be located. A 1 corresponds to the slot location of the logic analyzer.

`<module>` An integer, 1 through 10 (2 through 10 unused)

Examples

```
OUTPUT XXX;":INTERMODULE:INSERT 1,GROUP"
OUTPUT XXX;":INTERMODULE:INSERT OUT,1"
```

The following figure shows the result of the example output commands:



SKEW<N>

Command

```
:SKEW<N> <setting>
```

The SKEW command sets the skew value for a module. The <N> index value is the module number (1 corresponds to the logic analyzer and 2 through 10 unused). The <setting> parameter is the skew setting (– 1.0 to 1.0) in seconds.

`<N>` An integer, 1 through 10 (2 through 10 unused)

`<setting>` A real number from –1.0 to 1.0 seconds

Example

```
OUTPUT XXX;":INTERMODULE:SKEW1 3.0E-9"
```

Query

```
:SKEW<N>?
```

The query returns the user defined skew setting.

Returned Format

```
[INTermodule:SKEW<N>] <setting><NL>
```

Example

```
OUTPUT XXX;":INTERMODULE:SKEW1?"
```

TREE

Command

```
:TREE <module>,<module>
```

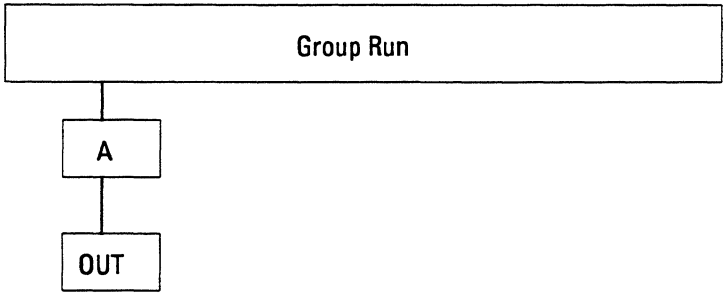
The TREE command allows an intermodule setup to be specified in one command. The first parameter is the intermodule arm value for module A (logic analyzer). The second parameter corresponds to the intermodule arm value for PORT OUT. A –1 means the module is not in the intermodule tree, a 0 value means the module is armed from the Intermodule run button (Group run), and a positive value indicates the module is being armed by another module with the slot location 1 to 10. A 1 corresponds to the slot location of the module A (logic analyzer) and 2 through 10 are unused.

<module> An integer, –1 through 10 (2 through 10 unused)

Example

```
OUTPUT XXX;":INTERMODULE:TREE 1,0"
```

The following figure shows the result of the example output commands:



Query

```
:TREE?
```

The TREE? query returns a string that represents the intermodule tree. A −1 means the module is not in the intermodule tree, a 0 value means the module is armed from the Intermodule run button (Group run), and a positive value indicates the module is being armed by another module with the slot location 1 to 10. A 1 corresponds to the slot location of the module A (logic analyzer) and 2 through 10 are unused.

Returned Format

```
[INTermodule:TREE] <module>,<module><NL>
```

Example

```
OUTPUT XXX;":INTERMODULE:TREE?"
```

TTIMe

Query

```
:TTIMe?
```

The TTIMe query returns values representing the absolute intermodule trigger time for all of the modules in the Intermodule configuration. Since the HP 1660A-series are single module instruments, this query is described only as a reminder that HP 16500A logic analysis system programs will run on the HP 1660A-series logic analyzers. The first value is the trigger time for the module in slot A, the second value is for the module in slot B, the third value is for slot C, etc.

The value 9.9E37 is returned when:

- The module in the corresponding slot is not time correlated; or
- A time correlatable module did not trigger.

> The trigger times returned by this command have already been offset by the INTermodule:SKEW values and internal hardware skews (INTermodule:HTIMe).

Returned Format `[:INTermodule:TTIMe] <value 1>,<value 2><NL>`

`<value 1>` Trigger time for module in slot A (real number)

`<value 2>` Trigger time for module in slot B (real number)

NOT USED

`<value10)` Trigger time for module in slot J (real number)

Example

```
OUTPUT XXX;":INTERMODULE:TTIME?"
```

13

MACHine Subsystem

Introduction

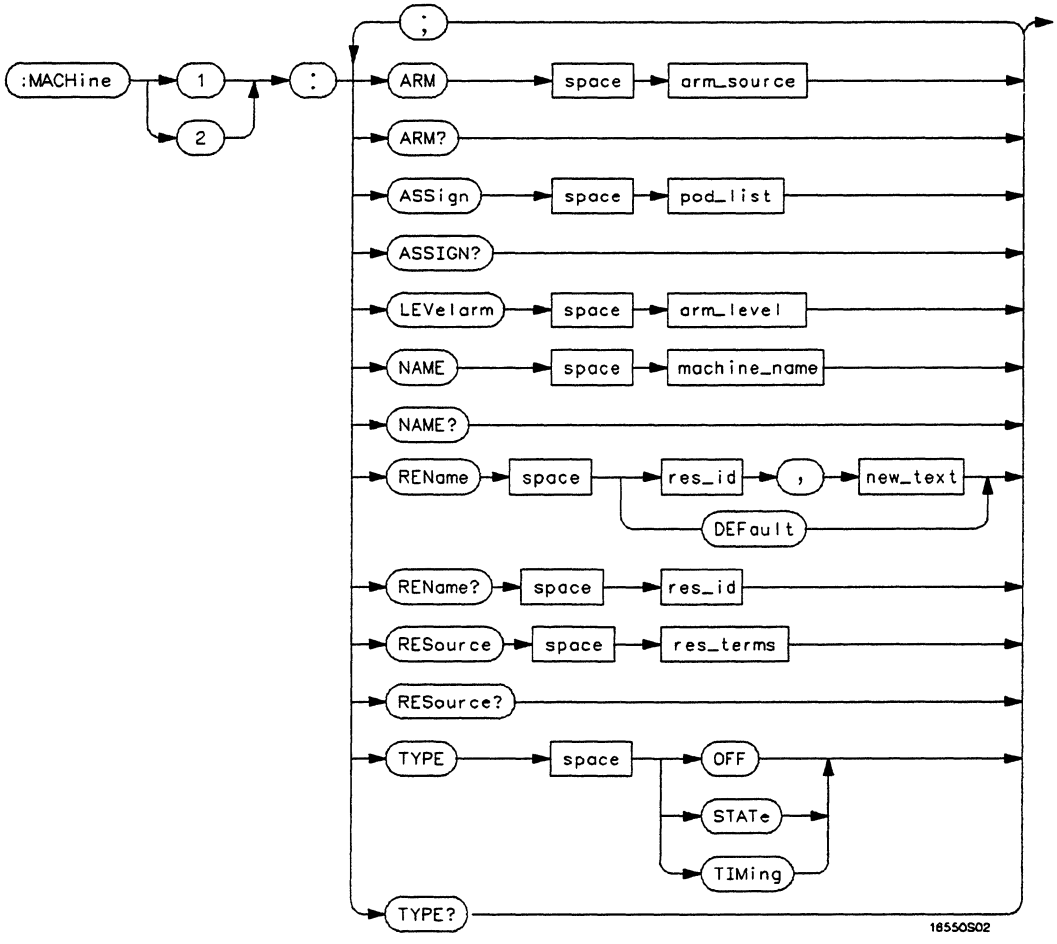
The MACHine subsystem contains the commands that control the machine level of operation of the logic analyzer. The functions of three of these commands reside in the State/Timing Configuration menu. These commands are:

- ASSign
- NAME
- TYPE

Even though the functions of the following commands reside in the Trace menu they are at the machine level of the command tree and are therefore located in the MACHine subsystem. These commands are:

- ARM
- LEVelarm
- REName
- RESource

Figure 13-1



Machine Subsystem Syntax Diagram

Table 13-1 **Machine Parameter Values**

Parameter	Values
arm_source	{RUN|INTermodule|MACHine{1|2}}
pod_list	{NONE|<pod num>[,<pod num>]...}
pod_num	{1|2|3|4|5|6|7|8}
arm_level	An integer from 1 to 11 representing sequence level
machine_name	A string of up to 10 alphanumeric characters
res_id	<state_terms> for state analyzer or {<state_terms>|GLEDge{1|2}} for timing analyzer
new_text	A string of up to 8 alphanumeric characters
state_terms	{A|B|C|D|E|F|G|H|I|J|RANGE{1|2}|TIMER{1|2}}
res_terms	{<res id>[,<res id>]...}

MACHine

Selector `:MACHine<N>`

The MACHine <N> selector specifies which of the two analyzers (machines) available in the HP 1660-series the commands or queries following will refer to. Because the MACHine<N> command is a root level command, it will normally appear as the first element of a compound header.

<N> {1|2} (the machine number)

Example

```
OUTPUT XXX; ":MACHINE1:NAME 'TIMING'"
```

ARM

Command

```
:MACHine{1|2}:ARM <arm_source>
```

The ARM command specifies the arming source of the specified analyzer (machine). The RUN option disables the arm source. For example, if you do not want to use either the intermodule bus or the other machine to arm the current machine, you specify the RUN option.

`<arm_source>` `{RUN|INTermodule|MACHine{1|2}}`

Example

```
OUTPUT XXX;":MACHINE1:ARM MACHINE2"
```

Query

```
:MACHine{1|2}:ARM?
```

The ARM query returns the source that the current analyzer (machine) wil be armed by.

Returned Format

```
[:MACHine{1|2}:ARM] <arm_source>
```

Example

```
OUTPUT XXX;":MACHINE:ARM?"
```

ASSign

Command

```
:MACHine{1|2}:ASSign <pod_list>
```

The ASSign command assigns pods to a particular analyzer (machine). The ASSign command will assign two pods for each pod number you specify because pods must be assigned to analyzers in pairs.

`<pod_list>` `{NONE|<pod >#[, <pod >#]...}`

`<pod>#` `{1|2|3|4|5|6|7|8}`

Example

```
OUTPUT XXX;":MACHINE1:ASSIGN 5, 2, 1"
```

Query

```
:MACHine{1|2}:ASSign?
```

The ASSign query returns which pods are assigned to the current analyzer (machine).

Returned Format

```
[:MACHine{1|2}:ASSign] <pod_list><NL>
```

<pod_list>

```
{NONE|<pod >#[, <pod >#]...}
```

<pod>#

```
{1|2|3|4|5|6|7|8}
```

Example

```
OUTPUT XXX;":MACHINE1:ASSIGN?"
```

LEVelarm

Command

```
:MACHine{1|2}:LEVelarm <arm_level>
```

The LEVelarm command allows you to specify the sequence level for a specified machine that will be armed by the Intermodule Bus or the other machine. This command is only valid if the specified machine is on and the arming source is not set to RUN with the ARM command.

<arm_level> An integer from 1 to the maximum number of levels specified in the appropriate trigger menu.

Example

```
OUTPUT XXX;":MACHINE1:LEVELARM 2"
```

Query

```
:MACHine{1|2}:LEVelarm?
```

The LEVelarm query returns the current sequence level receiving the arming for a specified machine.

Returned Format: `[:MACHine{1|2}:LEVelarm] <arm_level><NL>`

`<arm_level>` An integer from 1 to 11 representing sequence level

Example

```
OUTPUT XXX;":MACHINE1:LEVELARM?"
```

NAME

Command `:MACHine{1|2}:NAME <machine_name>`

The NAME command allows you to assign a name of up to 10 characters to a particular analyzer (machine) for easier identification.

`<machine_name>` A string of up to 10 alphanumeric characters

Example

```
OUTPUT XXX;":MACHINE1:NAME 'DRAMTEST'"
```

Query `:MACHine{1|2}:NAME?`

The NAME query returns the current analyzer name as an ASCII string.

Returned Format `[:MACHine{1|2}:NAME] <machine_name><NL>`

`<machine_name>` A string of up to 10 alphanumeric characters

Example

```
OUTPUT XXX;":MACHINE1:NAME?"
```

REName

Command

```
:MACHine{1|2}:REName {<res_id>, <new_text> |
DEFault}
```

The REName command allows you to assign a specific name of up to eight characters to terms A through J, Range 1 and 2, and Timer 1 and 2 in the state analyzer. In the timing analyzer, GLEDge (glitch/edge) 1 and 2 can be renamed in addition to the terms available in the state analyzer. The DEFault option sets all resource term names to the default names assigned when turning on the instrument.

`<res_id>` `<state_terms>` for state analyzer
or
`{<state_terms>|GLEDge{1|2}}` for timing analyzer

`<new_text>` A string of up to 8 alphanumeric characters

Example

```
OUTPUT XXX;":MACHINE1:RENAME A,'DATA'"
```

Query

```
:MACHine{1|2}:RENAME? <res_id>
```

The REName query returns the current names for specified terms assigned to the specified analyzer.

Returned Format

```
[:MACHine{1|2}:RENAME] <res_id>,<new_text><NL>
```

`<res_id>` `<state_terms>` for state analyzer
or
`{<state_terms>|GLEDge{1|2}}` for timing analyzer

`<new_text>` A string of up to 8 alphanumeric characters

Example

```
OUTPUT XXX;":MACHINE1:RENAME? D"
```

RESource

Command

```
:MACHine{1|2}:RESource <res_terms>
```

The RESource command allows you to assign resource terms A through J, Range 1 and 2, and Timer 1 and 2 to a particular analyzer (machine 1 or 2).

> In the timing analyzer only, two additional resource terms are available. These terms are GLEDge (Glitch/Edge) 1 and 2. These terms will always be assigned to the the machine that is configured as the timing analyzer.

`<res_terms>` `{A|B|C|D|E|F|G|H|I|J|TIMer1|TIMer2|RANGe1|RANGe2}`

Example

```
OUTPUT XXX;":MACHINE1:RESOURCE A,C,RANGE1"
```

Query

```
:MACHine{1|2}:RESOURCE?
```

The RESource query returns the current resource terms assigned to the specified analyzer.

Returned Format

```
[:MACHine{1|2}:RESOURCE] <res_terms>[,<res_terms>,...]<NL>
```

`<res_terms>` `{A|B|C|D|E|F|G|H|I|J|TIMer1|TIMer2|RANGe1|RANGe2}`

Example

```
OUTPUT XXX;":MACHINE1:RESOURCE?"
```

TYPE

Command `:MACHine{1|2}:TYPE <analyzer_type>`

The TYPE command specifies what type a specified analyzer (machine) will be. The analyzer types are state or timing. The TYPE command also allows you to turn off a particular machine.

> Only one timing analyzer can be specified at a time.

`<analyzer_type>` `{OFF|STATe|TIMing}`

Example

```
OUTPUT XXX;":MACHINE1:TYPE STATE"
```

Query `:MACHine{1|2}:TYPE?`

The TYPE query returns the current analyzer type for the specified analyzer.

Returned Format `[:MACHine{1|2}:TYPE] <analyzer_type><NL>`

`<analyzer_type>` `{OFF|STATe|TIMing}`

Example

```
OUTPUT XXX;":MACHINE1:TYPE?"
```

14

WLISt Subsystem

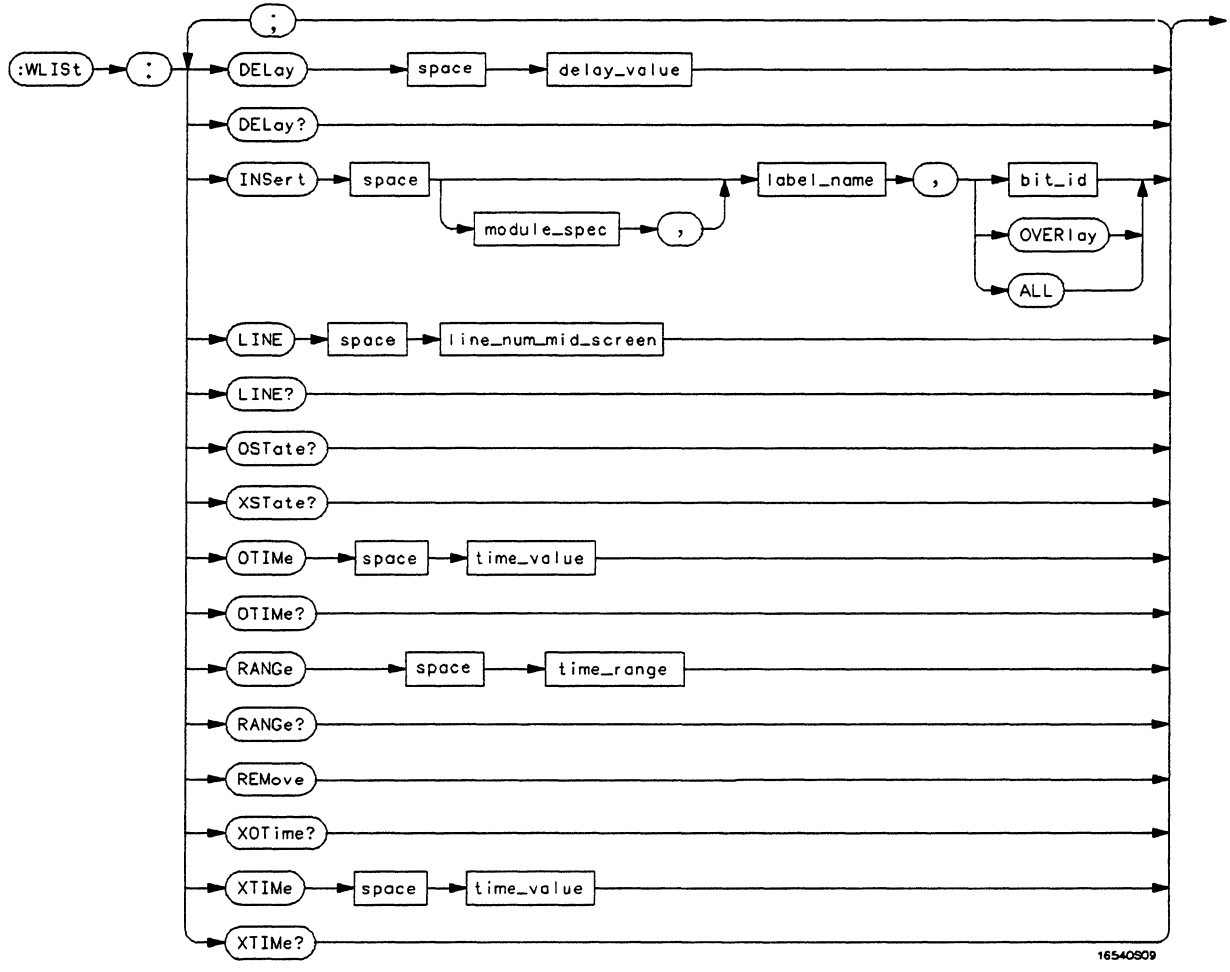
Introduction

The WLISt subsystem contains the commands available for the Timing/State mixed mode display. The X and O markers can only be placed on the waveforms in the waveform portion of the Timing/State mixed mode display. The XSTate and OSTate queries return what states the X and O markers are on. Because the markers can only be placed on the timing waveforms, the queries return what state (state acquisition memory location) the marked pattern is stored in.

In order to have mixed mode, one machine must be a state analyzer with time tagging on (use MACHine<N>:STRigger:TAG TIME).

- DELay
- INSert
- LINE
- OSTate
- OTIMe
- RANGe
- REMove
- XOTime
- XSTate
- XTIMe

Figure 14-1



WLISt Subsystem Syntax Diagram

Table 14-1 **WLISt Parameter Values**

Parameter	Value
delay_value	Real number between −2500 s and +2500 s
module_spec	{1|2|3|4|5|6|7|8|9|10} (slot where timing card is installed, 2 through 10 unused)
bit_id	An integer from 0 to 31
label_name	String of up to 6 alphanumeric characters
line_num_mid_screen	An integer from −8191 to +8191
waveform	String containing <acquisition spec>{1|2}
time_value	Real number
time_range	Real number between 10 ns and 10 ks

WLISt

Selector

```
:WLISt
```

The WLISt (Waveforms/LISting) selector is used as a part of a compound header to access the settings normally found in the Mixed Mode menu. Because the WLISt command is a root level command, it will always appear as the first element of a compound header.

> The WLISt subsystem is only available when one or more state analyzers, with time tagging on, are specified.

Example

```
OUTPUT XXX;":WLIST:XTIME 40.0E-6"
```

DELay

Command `:MACHine{1|2}:WLISt:DELay <delay_value>`

The DELay command specifies the amount of time between the timing trigger and the horizontal center of the the timing waveform display. The allowable values for delay are −2500 s to +2500 s.

`<delay_value>` Real number between −2500 s and +2500 s

Example

```
OUTPUT XXX;":MACHINE1:WLIST:DELAY 100E-6"
```

Query `:MACHine{1|2}:WLISt:DELay?`

The DELay query returns the current time offset (delay) value from the trigger.

Returned Format `[:MACHine{1|2}:WLISt:DELay] <time_value><NL>`

`<delay_value>` Real number between −2500 s and +2500 s

Example

```
OUTPUT XXX;":MACHINE1:WLIST:DELAY?"
```

INSert

Command

```
:MACHine{1|2}:WLISt:INSert [<module_spec>,]
<label_name>[,{<bit_id>|OVERlay|ALL}]
```

The INSert command inserts waveforms in the timing waveform display. The waveforms are added from top to bottom up to a maximum of 96 waveforms. Once 96 waveforms are present, each time you insert another waveform, it replaces the last waveform.

The first parameter specifies from which module the waveform is coming from; however, the HP 1660A-series logic analyzers are single-module instruments. Therefore, this parameter is not needed. It is described here as a reminder that programs for the HP 16500A logic analysis system can be used. The second parameter specifies the label name that will be inserted. The optional third parameter specifies the label bit number, overlay, or all. If a number is specified, only the waveform for that bit number is added to the screen.

If you specify OVERlay, all the bits of the label are displayed as a composite overlaid waveform. If you specify ALL, all the bits are displayed sequentially. If you do not specify the third parameter, ALL is assumed.

`<module_spec>` {1|2|3|4|5|6|7|8|9|10} (not needed)

`<label_name>` String of up to 6 alphanumeric characters

`<bit_id>` An integer from 0 to 31

Examples

Inserting a logic analyzer waveform:

```
OUTPUT XXX;":MACHINE1:WLIST:INSERT 3, 'WAVE',10"
```

LINE

Command

```
:MACHine{1|2}:WLISt:LINE <line_num_mid_screen>
```

The LINE command allows you to scroll the state analyzer listing vertically. The command specifies the state line number relative to the trigger that the analyzer highlights at the center of the screen.

<line_num_mid_screen> An integer from −8191 to +8191

Example

```
OUTPUT XXX;":MACHINE1:WLIST:LINE 0"
```

Query

```
:MACHine{1|2}:WLISt:LINE?
```

The LINE query returns the line number for the state currently in the box at center screen.

Returned Format

```
[:MACHine{1|2}:WLISt:LINE] <line_num_mid_screen><NL>
```

Example

```
OUTPUT XXX;":MACHINE1:WLIST:LINE?"
```

OSTate

Query `:WLISt:OSTate?`

The OSTate query returns the state where the O Marker is positioned. If data is not valid, the query returns 32767.

Returned Format `[:WLISt:OSTate] <state_num><NL>`

`<state_num>` An integer from −8191 to +8191

Example

```
OUTPUT XXX;":WLIST:OSTATE?"
```

OTIMe

Command `:WLISt:OTIMe <time_value>`

The OTIMe command positions the O Marker on the timing waveforms in the mixed mode display. If the data is not valid, the command performs no action.

`<time_value>` A real number

Example

```
OUTPUT XXX;":WLIST:OTIME 40.0E−6"
```

Query `:WLISt:OTIMe?`

The OTIMe query returns the O Marker position in time. If data is not valid, the query returns 9.9E37.

Returned Format `[:WLISt:OTIMe] <time_value><NL>`

`<time_value>` A real number

Example

```
OUTPUT XXX;":WLIST:OTIME?"
```

RANGe

Command `:MACHine{1|2}:WLISt:RANGe <time_value>`

The RANGe command specifies the full-screen time in the timing waveform menu. It is equivalent to ten times the seconds per division setting on the display. The allowable values for RANGe are from 10 ns to 10 ks.

`<time_value>` A real number between 10 ns and 10 ks

Example

```
OUTPUT XXX;":MACHINE1:WLIST:RANGE 100E-9"
```

Query `:MACHine{1|2}:WLISt:RANGe?`

The RANGe query returns the current full-screen time.

Returned Format `[:MACHine{1|2}:WLISt:RANGe] <time_value><NL>`

`<time_value>` A real number between 10 ns and 10 ks

Example

```
OUTPUT XXX;":MACHINE1:WLIST:RANGE?"
```

REMove

Command

```
:MACHine{1|2}:WLISt:REMove
```

The REMove command deletes all waveforms from the display.

Example

```
OUTPUT XXX;":MACHINE1:WLIST:REMOVE"
```

XOTime

Query

```
:MACHine{1|2}:WLISt:XOTime?
```

The XOTime query returns the time from the X marker to the O marker. If data is not valid, the query returns 9.9E37.

Returned Format

```
[:MACHine{1|2}:WLISt:XOTime] <time_value><NL>
```

`<time_value>` A real number

Example

```
OUTPUT XXX;":MACHINE1:WLIST:XOTIME?"
```

XSTate

Query `:WLISt:XSTate?`

The XSTate query returns the state where the X Marker is positioned. If data is not valid, the query returns 32767.

Returned Format `[:WLISt:XSTate] <state_num><NL>`

`<state_num>` An integer

Example

```
OUTPUT XXX;":WLIST:XSTATE?"
```

XTIMe

Command `:WLISt:XTIMe <time_value>`

The XTIMe command positions the X Marker on the timing waveforms in the mixed mode display. If the data is not valid, the command performs no action.

`<time_value>` A real number

Example

```
OUTPUT XXX;":WLIST:XTIME 40.0E-6"
```

Query `:WLISt:XTIMe?`

The XTIMe query returns the X Marker position in time. If data is not valid, the query returns 9.9E37.

Returned Format `[:WLISt:XTIMe] <time_value><NL>`

`<time_value>` A real number

Example

```
OUTPUT XXX;":WLIST:XTIME?"
```

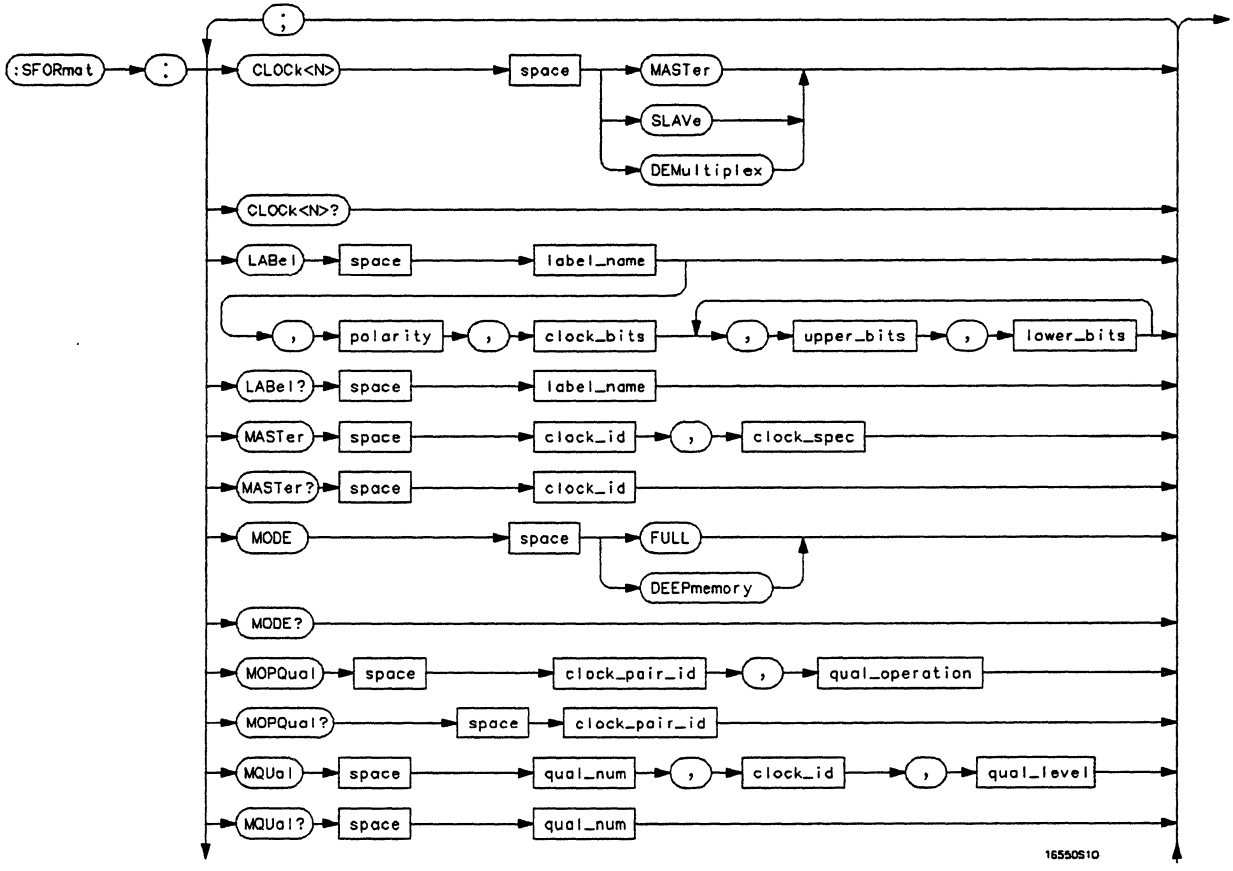
15

SFORmat Subsystem

The SFORmat subsystem contains the commands available for the State Format menu in the HP 1660A-series logic analyzers. These commands are:

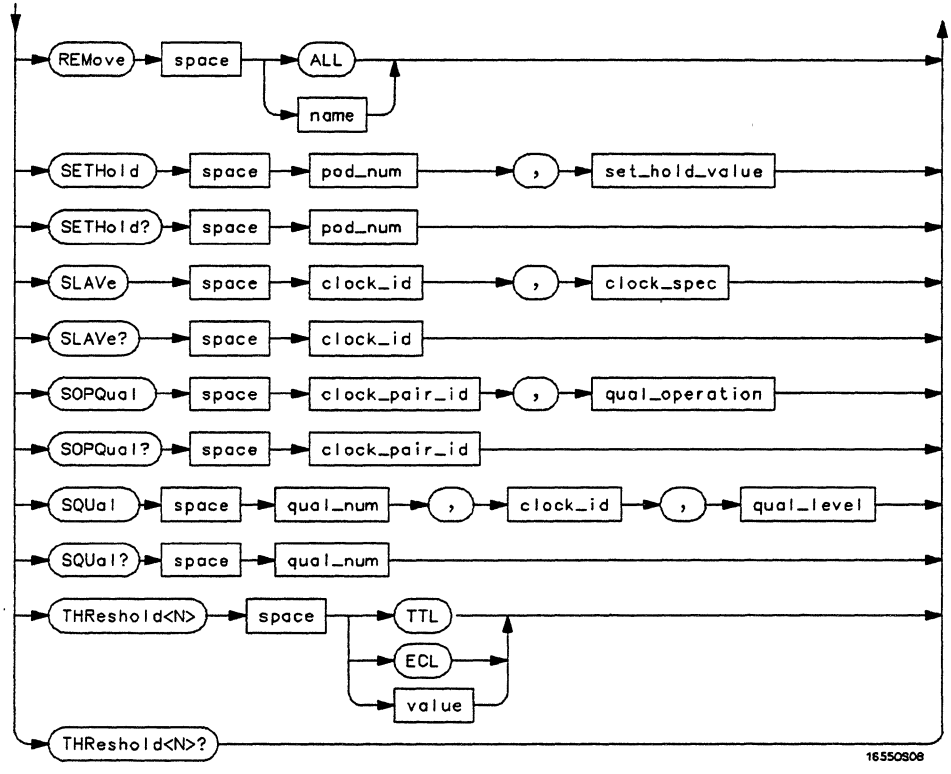
- CLOCk
- LABel
- MASTer
- MODE
- MOPQual
- MQUal
- REMove
- SETHold
- SLAVe
- SOPQual
- SQUal
- THReshold

Figure 15-1



SFORmat Subsystem Syntax Diagram

Figure 15-1



SFORmat Subsystem Syntax Diagram (continued)

Table 15-1 **SFORmat Parameter Values**

Parameter	Values
<N>	{{1|2}|{3|4|5|6}|{7|8}}
label_name	String of up to 6 alphanumeric characters
polarity	{POSitive|NEGative}
clock_bits	Format (integer from 0 to 63) for a clock (clocks are assigned in decreasing order)
upper_bits	Format (integer from 0 to 65535) for a pod (pods are assigned in decreasing order)
lower_bits	Format (integer from 0 to 65535) for a pod (pods are assigned in decreasing order)
clock_id	{J|K|L|M|N|P}
clock_spec	{OFF|RISing|FALLing|BOTH}
clock_pair_id	{1|2}
qual_operation	{AND|OR}
qual_num	{1|2|3|4}
qual_level	{OFF|LOW|HIGH}
pod_num	{1|2|3|4|5|6|7|8}
set_hold_value	{0|1|2|3|4|5|6|7|8|9}
value	voltage (real number) −6.00 to +6.00

SFORmat

Selector

```
:MACHine{1|2}:SFORmat
```

The SFORmat (State Format) selector is used as a part of a compound header to access the settings in the State Format menu. It always follows the MACHine selector because it selects a branch directly below the MACHine level in the command tree.

Example

```
OUTPUT XXX;":MACHINE2:SFORMAT:MASTER J, RISING"
```

CLOCk

Command

```
:MACHine{1|2}:SFORmat:CLOCk<N> <clock_mode>
```

The CLOCk command selects the clocking mode for a given pod when the pod is assigned to the state analyzer. When the MASTer option is specified, the pod will sample all 16 channels on the master clock. When the SLAVe option is specified, the pod will sample all 16 channels on the slave clock. When the DEMultiplex option is specified, only one pod of a pod pair can acquire data. The 16 bits of the selected pod will be clocked by the demultiplex master for labels with bits assigned under the Master pod. The same 16 bits will be clocked by the demultiplex slave for labels with bits assigned under the Slave pod. The master clock always follows the slave clock when both are used.

`<N>` `{{1|2}| {3|4}|{5|6}|{7|8}}` 1 through 8 for the HP 1660A, 1 through 6 for the HP 1661A, 1 through 4 for the HP 1662A, and 1 through 2 for the HP 1663A.

`<clock_mode>` `{MASTer|SLAVe|DEMultiplex}`

Example

```
OUTPUT XXX;":MACHINE1:SFORMAT:CLOCK2 MASTER"
```

Query `:MACHine{1|2}:SFORmat:CLOCk<N>?`

The CLOCk query returns the current clocking mode for a given pod.

Returned Format `[:MACHine{1|2}:SFORmat:CLOCK<N>] <clock_mode><NL>`

Example

```
OUTPUT XXX; ":MACHINE1:SFORMAT:CLOCK2?"
```

LABel

Command `:MACHine{1|2}:SFORmat:LABel <name>,[<polarity>, <clock_bits>, <upper_bits>, <lower_bits> [,<upper_bits>,<lower_bits>]...]`

The LABel command allows you to specify polarity and assign channels to new or existing labels. If the specified label name does not match an existing label name, a new label will be created.

The order of the pod-specification parameters is significant. The first one listed will match the highest numbered pod assigned to the machine you're using. Each pod specification after that is assigned to the next highest numbered pod. This way they match the left-to-right descending order of the pods you see on the Format display. Not including enough pod specifications results in the lowest numbered pod(s) being assigned a value of zero (all channels excluded). If you include more pod specifications than there are pods for that machine, the extra ones will be ignored. However, an error is reported anytime when more than 13 pod specifications are listed.

The polarity can be specified at any point after the label name.

Because pods contain 16 channels, the format value for a pod must be between 0 and 65535 ($2^{16}-1$). When giving the pod assignment in binary (base 2), each bit will correspond to a single channel. A "1" in a bit position means the associated channel in that pod is assigned to that pod and bit. A "0" in a bit position means the associated channel in that pod is excluded from the label. For example, assigning #B1111001100 is equivalent to entering "....****..**.." from the front panel.

A label can not have a total of more than 32 channels assigned to it.

`<name>` String of up to 6 alphanumeric characters

`<polarity>` `{POSitive|NEGative}`

`<clock_bits>` Format (integer from 0 to 63) for a clock (clocks are assigned in decreasing order)

`<upper_bits>` Format (integer from 0 to 65535) for a pod (pods are assigned in decreasing order)

`<lower_bits>` Format (integer from 0 to 65535) for a pod (pods are assigned in decreasing order)

Examples

```
OUTPUT XXX;":MACHINE2:SFORMAT:LABEL 'STAT', POSITIVE,
0,127,40312"
OUTPUT XXX;":MACHINE2:SFORMAT:LABEL 'SIG 1', #B11,
#B0000000011111111,#B0000000000000000 "
```

Query

```
:MACHine{1|2}:SFORmat:LABel? <name>
```

The LABel query returns the current specification for the selected (by name) label. If the label does not exist, nothing is returned. The polarity is always returned as the first parameter. Numbers are always returned in decimal format.

Returned Format

```
[:MACHine{1|2}:SFORmat:LABel] <name>,<polarity>
[, <clock_bits>, <upper_bits>, <lower_bits>]<NL>
```

Example

```
OUTPUT XXX;":MACHINE2:SFORMAT:LABEL? 'DATA'"
```

MASTer

Command Syntax: `:MACHine{1|2}:SFORmat:MASTer <clock_id>, <clock_spec>`

The MASTer clock command allows you to specify a master clock for a given machine. The master clock is used in all clocking modes (Master, Slave, and Demultiplexed). Each command deals with only one clock (J,K,L,M,N,P); therefore, a complete clock specification requires six commands, one for each clock. Edge specifications (RISing, FALLing, or BOTH) are ORed.

> At least one clock edge must be specified.

`<clock_id>` `{J|K|L|M|N|P}`

`<clock_spec>` `{OFF|RISing|FALLing|BOTH}`

Example

```
OUTPUT XXX;":MACHINE2:SFORMAT:MASTER J, RISING"
```

Query `:MACHine{1|2}:SFORmat:MASTer? <clock_id>`

The MASTer query returns the clock specification for the specified clock.

Returned Format `[:MACHine{1|2}:SFORmat:MASTer]<clock_id>,<clock_spec><NL>`

Example

```
OUTPUT XXX;":MACHINE2:SFORMAT:MASTER? <clock_id>"
```

MODE

Command

```
:MACHine{1|2}:SFORmat:MODE <acq_mode>
```

The MODE command allows you to select the acquistion mode of the state analyzer. The modes are either full-channel with 4 Kbit of memory depth per channel or half-channel with 8 Kbit of memory depth per channel.

`<acq_mode>` `{FULL|DEEPmemory}`

Example

```
OUTPUT XXX;":MACHine1:SFORMAT:MODE FULL"
```

Query

```
:MACHine{1|2}:SFORmat:MODE?
```

The MODE query returns the current acquistion mode.

Returned Format

```
[:MACHine{1|2}:SFORmat:MODE] <acq_mode><NL>
```

Example

```
OUTPUT XXX;":MACHINE1:SFORMAT:MODE?"
```

MOPQual

Command

```
:MACHine{1|2}:SFORmat:MOPQual <clock_pair_id>,
<qual_operation>
```

The MOPQual (master operation qualifier) command allows you to specify either the AND or the OR operation between master clock qualifier pair 1 and 2, or between master clock qualifier pair 3 and 4. For example, you can specify a master clock operation qualifer 1 AND 2.

`<clock_pair_id>` `{1|2}`

`<qual_operation>` `{AND|OR}`

Example

```
OUTPUT XXX;":MACHINE1:SFORMAT:MOPQUAL 1,AND"
```

Query

```
:MACHine{1|2}:SFORmat:MOPQUal? <clock_pair_id>
```

The MOPQual query returns the operation qualifier specified for the master clock.

Returned Format:

```
[:MACHine{1|2}:SFORmat:MOPQUal <clock_pair_id>]
<qual_operation><NL>
```

Example

```
OUTPUT XXX;":MACHine1:SFORMAT:MOPQUAL? 1"
```

MQUal

Command

```
:MACHine{1|2}:SFORmat:MQUal
<qual_num>,<clock_id>,<qual_level>
```

The MQUal (master qualifier) command allows you to specify the level qualifier for the master clock.

`<qual_num>` `{{1|2}|{3|4}}` 1 through 4 for HP 1660A, HP 1661A, HP 1662A; or, 1 or 2 for HP 1663A.

`<clock_id>` `{J|K|L|M|N|P}`

`<qual_level>` `{OFF|LOW|HIGH}`

Example

```
OUTPUT XXX;":MACHINE2:SFORMAT:MQUAL 1,J,LOW"
```

Query

```
:MACHine{1|2}:SFORmat:MQUal? <qual num>
```

The MQUal query returns the qualifier specified for the master clock.

Returned Format

```
[:MACHine{1|2}:SFORmat:MQUal] <qual_level><NL>
```

Example

```
OUTPUT XXX;":MACHINE2:SFORMAT:MQUAL? 1"
```

REMove

Command

```
:MACHine{1|2}:SFORmat:REMove {<name>|ALL}
```

The REMove command allows you to delete all labels or any one label for a given machine.

`<name>` String of up to 6 alphanumeric characters

Examples

```
OUTPUT XXX;":MACHINE2:SFORMAT:REMOVE 'A'"
OUTPUT XXX;":MACHINE2:SFORMAT:REMOVE ALL"
```

SETHold

Command

```
:MACHine{1|2}:SFORmat:SETHold <pod_num>,
<set_hold_value>
```

The SETHold (setup/hold) command allows you to set the setup and hold specification for the state analyzer.

> Even though the command requires integers to specify the setup and hold, the query returns the current settings in a string. For example, if you send the integer 0 for the setup and hold value, the query will return 3.5/0.0 ns as an ASCII string when you have one clock and one edge specified.

`<pod_num>` `{{1|2}|{3|4}|{5|6}|{7|8}}` 1 through 8 for the HP 1660A, 1 through 6 for the HP 1661A, 1 through 4 for the HP 1662A, and 1 through 2 for the HP 1663A.

`<set_hold_value>` An integer `{0|1|2|3|4|5|6|7|8|9}` representing the setup and hold values in table 15-2.

Table 15-2 **Setup and hold values**

For one clock and one edge	For one clock and both edges	For multiple clocks
0 = 3.5/0.0 ns	0 = 4.0/0.0	0 = 4.5/0.0
1 = 3.0/0.5 ns	1 = 3.5/0.5	1 = 4.0/0.5
2 = 2.5/1.0 ns	2 = 3.0/1.0	2 = 3.5/1.0
3 = 2.0/1.5 ns	3 = 2.5/1.5	3 = 3.0/1.5
4 = 1.5/2.0 ns	4 = 2.0/2.0	4 = 2.5/2.0
5 = 1.0/2.5 ns	5 = 1.5/2.5	5 = 2.0/2.5
6 = 0.5/3.0 ns	6 = 1.0/3.0	6 = 1.5/3.0
7 = 0.0/3.5 ns	7 = 0.5/3.5	7 = 1.0/3.5
N/A	8 = 0.0/4.0	8 = 0.5/4.0
N/A	N/A	9 = 0.0/4.5

Example

```
OUTPUT XXX;":MACHINE2:SFORMAT:SETHOLD 1,2"
```

Query

```
:MACHine{1|2}:SFORMAT:SETHOLD? <pod_num>
```

The SETHold query returns the current setup and hold settings.

Returned Format

```
[:MACHine{1|2}:SFORmat:SETHold <pod_num>] <set_hold_value><NL>
```

Example

```
OUTPUT XXX;":MACHINE2:SFORMAT:SETHOLD? 3"
```

SLAVe

Command

```
:MACHine{1|2}:SFORmat:SLAVe <clock_id>,<clock_spec>
```

The SLAVe clock command allows you to specify a slave clock for a given machine. The slave clock is only used in the Slave and Demultiplexed clocking modes. Each command deals with only one clock (J,K,L,M,N,P); therefore, a complete clock specification requires six commands, one for each clock. Edge specifications (RISing, FALLing, or BOTH) are ORed.

> When slave clock is being used at least one edge must be specified.

`<clock_id>` `{J|K|L|M|N|P}`

`<clock_spec>` `{OFF|RISing|FALLing|BOTH}`

Example

```
OUTPUT XXX;":MACHINE2:SFORMAT:SLAVE J, RISING"
```

Query

```
:MACHine{1|2}:SFORmat:SLAVe?<clock id>
```

The SLAVe query returns the clock specification for the specified clock.

Returned Format

```
[:MACHine{1|2}:SFORmat:SLAVe] <clock_id>,<clock_spec><NL>
```

Example

```
OUTPUT XXX;":MACHINE2:SFORMAT:SLAVE? K"
```

SOPQual

Command

```
:MACHine{1|2}:SFORmat:SOPQual <clock_pair_id>,
<qual operation>
```

The SOPQual (slave operation qualifier) command allows you to specify either the AND or the OR operation between slave clock qualifier pair 1 and 2, or between slave clock qualifier pair 3 and 4. For example you can specify a slave clock operation qualifer 1 AND 2.

`<clock_pair_id>` `{1|2}`

`<qual_operation>` `{AND|OR}`

Example

```
OUTPUT XXX;":MACHine2:SFORMAT:SOPQUAL 1,AND"
```

Query

```
:MACHine{1|2}:SFORmat:SOPQual? <clock pair id>
```

The SOPQual query returns the operation qualifier specified for the slave clock.

Returned Format

```
[:MACHine{1|2}:SFORmat:SOPQual <clock_pair_id>]
<qual_operation><NL>
```

Example

```
OUTPUT XXX;":MACHiNE2:SFORMAT:SOPQUAL? 1"
```

SQUal

Command

```
:MACHine{1|2}:SFORmat:SQUal <qual_num>,<clock_id>,
<qual_level>
```

The SQUal (slave qualifier) command allows you to specify the level qualifier for the slave clock.

`<qual_num>` `{{1|2}|{3|4}}` 1 through 4 for HP 1660A, HP 1661A, HP 1662A; or, 1 or 2 for HP 1663A.

`<clock_id>` `{J|K|L|M|N|P}`

`<qual_level>` `{OFF|LOW|HIGH}`

Example

```
OUTPUT XXX;":MACHINE2:SFORMAT:SQUAL 1,J,LOW"
```

Query

```
:MACHine{1|2}:SFORmat:SQUal?<qual_num>
```

The SQUal query returns the qualifier specified for the slave clock.

Returned Format

```
[:MACHine{1|2}:SFORmat:SQUal] <clock_id>,<qual_level><NL>
```

Example

```
OUTPUT XXX;":MACHINE2:SFORMAT:SQUAL? 1"
```

THReshold

Command

```
:MACHine{1|2}:SFORmat:THReshold<N>
{TTL|ECL|<value>}
```

The THReshold command allows you to set the voltage threshold for a given pod to ECL, TTL, or a specific voltage from −6.00 V to +6.00 V in 0.05 volt increments.

`<N>` `{{1|2}|{3|4}|{5|6}|{7|8}}` 1 through 8 for the HP 1660A, 1 through 6 for the HP 1661A, 1 through 4 for the HP 1662A, and 1 through 2 for the HP 1663A.

`<value>` Voltage (real number) −6.00 to +6.00

`TTL` Default value of +1.6 V

`ECL` Default value of −1.3 V

Example

```
OUTPUT XXX;":MACHINE1:SFORMAT:THRESHOLD1 4.0"
```

Query

```
:MACHine{1|2}:SFORmat:THReshold<N>?
```

The THReshold query returns the current threshold for a given pod.

Returned Format

```
[:MACHine{1|2}:SFORmat:THReshold<N>] <value><NL>
```

Example

```
OUTPUT XXX;":MACHINE1:SFORMAT:THRESHOLD4?"
```

16

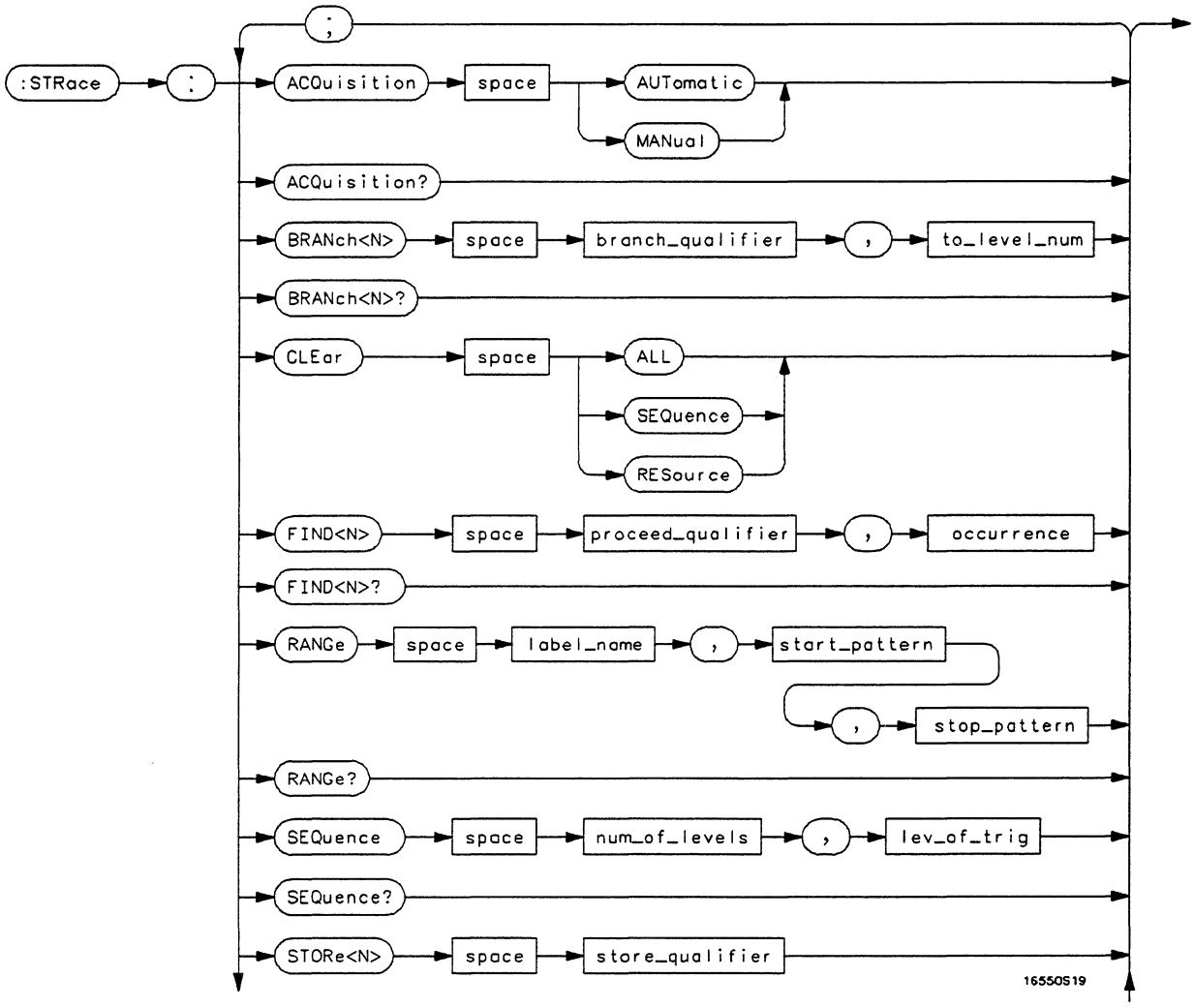
STRigger (STRace) Subsystem

Introduction

The STRigger subsystem contains the commands available for the State Trigger menu in the HP 1660A-series logic analyzers. The State Trigger subsystem will also accept the STRace Command as used in previous HP 1650-series logic analyzers to eliminate the need to rewrite programs containing STRace as the Command keyword. The STRigger subsystem commands are:

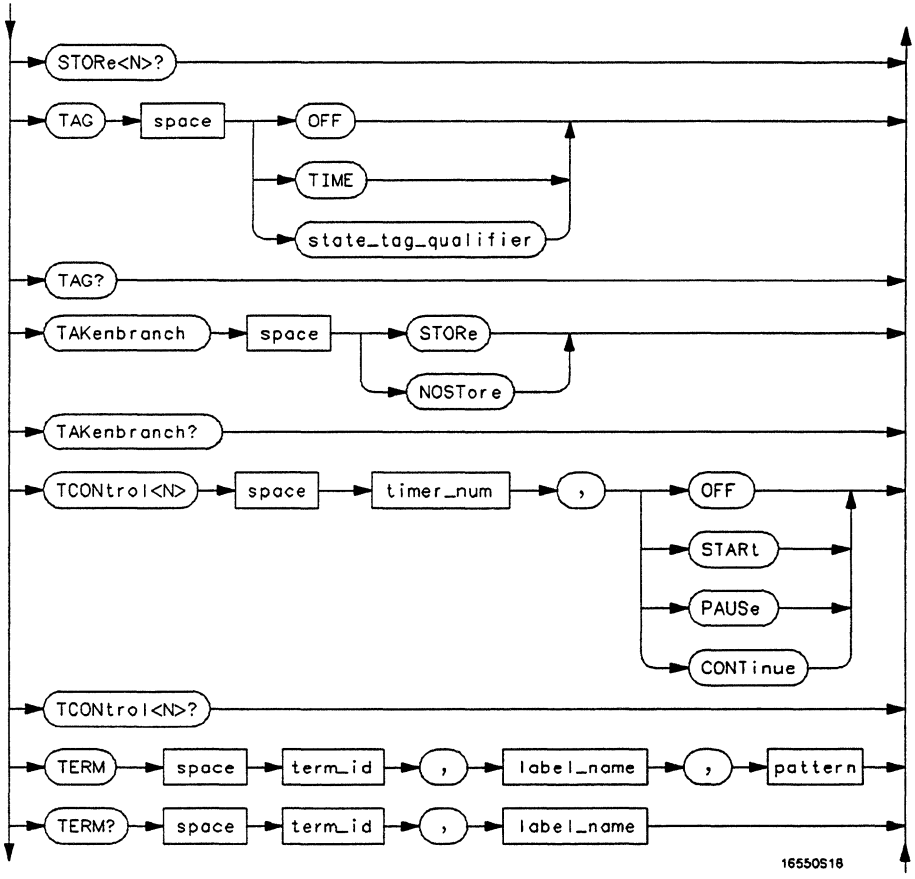
- ACQuisition
- BRANch
- CLEar
- FIND
- RANGe
- SEQuence
- STORe
- TAG
- TAKenbranch
- TCONtrol
- TERM
- TIMER
- TPOSition

Figure 16-1



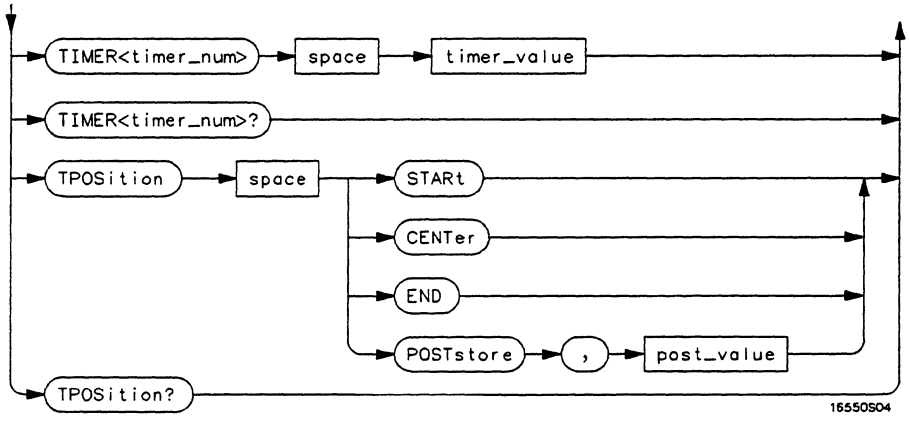
STRigger Subsystem Syntax Diagram

Figure 16-1 (continued)



STRigger Subsystem Syntax Diagram (continued)

Figure 16-1 (continued)



STRigger Subsystem Suntax Diagram (continued)

Table 16-1 **STRigger Parameter Values**

Parameter	Values
branch_qualifier	<qualifier>
to_lev_num	integer from 1 to last level
proceed_qualifier	<qualifier>
occurrence	number from 1 to 1048575
label_name	string of up to 6 alphanumeric characters
start_pattern	"{#B{0|1}...| #Q{0|1|2|3|4|5|6|7}...| #H{0|1|2|3|4|5|6|7|8|9|A|B|C|D|E|F}...| {0|1|2|3|4|5|6|7|8|9}...}"
stop_pattern	"{#B{0|1}...| #Q{0|1|2|3|4|5|6|7}...| #H{0|1|2|3|4|5|6|7|8|9|A|B|C|D|E|F}...| {0|1|2|3|4|5|6|7|8|9}...}"
num_of_levels	integer from 2 to 12
lev_of_trig	integer from 1 to (number of existing sequence levels — 1)
store_qualifier	<qualifier>
state_tag_qualifier	<qualifier>
timer_num	{1|2}
timer_value	400 ns to 500 seconds
term_id	{A|B|C|D|E|F|G|H|I|J}
pattern	"{#B{0|1|X}...| #Q{0|1|2|3|4|5|6|7|x}...| #H{0|1|2|3|4|5|6|7|8|9|A|B|C|D|E|F|x}...| {0|1|2|3|4|5|6|7|8|9}...}"
qualifier	see "Qualifier" on page 16-7
post_value	integer from 0 to 100 representing percentage

Qualifier

The qualifier for the state trigger subsystem can be terms A through J, Timer 1 and 2, and Range 1 and 2. In addition, qualifiers can be the NOT boolean function of terms, timers, and ranges. The qualifier can also be an expression or combination of expressions as shown below and figure 16-2, "Complex Qualifier," on page 16-11.

The following parameters show how qualifiers are specified in all commands of the STRigger subsystem that use `<qualifier>`.

```
<qualifier>     {"ANYSTATE"|"NOSTATE"|"<expression>"}

<expression>    {<expression1a>|<expression1b>|<expression1a> OR
                <expression1b>|<expression1a> AND <expression1b>}

<expression1a>  {<expression1a_term>|(<expression1a_term>[ OR
                <expression1a_term>]* )|(<expression1a_term>[ AND
                <expression1a_term>]* )}

<expression1a_  { <expression2a>|<expression2b>|<expression2c>|<expression2d>}
term>

<expression1b>  {<expression1b_term>|( <expression1b_term>[ OR
                <expression1b_term>]* )|(<expression1b_term>[ AND
                <expression1b_term>]* )}

<expression1b_  {<expression2e>|<expression2f>|<expression2g>|<expression2h>}
term>

<expression2a>  {<term3a>|<term3b>|(<term3a> <boolean_op> <term3b>)}

<expression2b>  {<term3c>|<range3a>|(<term3c> <boolean_op> <range3a>)}

<expression2c>  {<term3d>}

<expression2d>  {<term3e>|<timer3a>|(<term3e> <boolean_op> <timer3a>)}

<expression2e>  {<term3f>|<term3g>|(<term3f> <boolean_op> <term3g>)}

<expression2f>  {<term3h>|<range3b>|(<term3h> <boolean_op> <range3b>)}

<expression2g>  {<term3i>}

<expression2h>  {<term3j>|<timer3b>|(<term3e> <boolean_op> <timer3b>)}

<boolean_op>    {AND|NAND|OR|NOR|XOR|NXOR}

<term3a>        {A|NOTA}
```

<term3b>	{B|NOTB}
<term3c>	{C|NOTC}
<term3d>	{D|NOTD}
<term3e>	{E|NOTE}
<term3f>	{F|NOTF}
<term3g>	{G|NOTG}
<term3h>	{H|NOTH}
<term3i>	{I|NOTI}
<term3j>	{J|NOTJ}
<range3a>	{IN_RANGE1|OUT_RANGE1}
<range3b>	{IN_RANGE2|OUT_RANGE2}
<timer3a>	{TIMER1<|TIMER1>}
<timer3b>	{TIMER2<|TIMER2>}

Qualifier Rules

The following rules apply to qualifiers:

- Qualifiers are quoted strings and, therefore, need quotes.
- Expressions are evaluated from left to right.
- Parenthesis are used to change the order evaluation and, therefore, are optional.
- An expression must map into the combination logic presented in the combination pop-up menu within the STRigger menu (see figure 16-2 on page 16-12).

Examples

```
'A'
'( A OR B )'
'(( A OR B ) AND C )'
'(( A OR B ) AND C AND IN_RANGE2 )'
'(( A OR B ) AND ( C AND IN_RANGE1 ))'
'IN_RANGE1 AND ( A OR B ) AND C'
```

STRigger (STRace)

Selector

```
:MACHine{1|2}:STRigger
```

The STRigger (STRace) (State Trigger) Command is used as a part of a compound header to access the settings found in the State Trace menu. It always follows the MACHine Command because it selects a branch directly below the MACHine level in the command tree.

Example

```
OUTPUT XXX;":MACHINE1:STRIGGER:TAG TIME"
```

ACQuisition

Command

```
:MACHine{1|2}:STRigger:ACQuisition
{AUTOmatic|MANual}
```

The ACQuisition command allows you to specify the acquisition mode for the State analyzer.

Example

```
OUTPUT XXX;":MACHINE1:STRIGGER:ACQUISITION AUTOMATIC"
```

Query

```
:MACHine{1|2}:STRigger:ACQuisition?
```

The ACQuisition query returns the current acquisition mode specified.

Returned Format

```
[:MACHine{1|2}:STRigger:ACQuisition] {AUTOmatic|MANual}<NL>
```

Example

```
OUTPUT XXX;":MACHINE1:STRIGGER:ACQUISITION?"
```

BRANch

Command

```
:MACHine{1|2}:STRigger:BRANch<N>
<branch_qualifier>,<to_level_number>
```

The BRANch command defines the branch qualifier for a given sequence level. When this branch qualifier is matched, it will cause the sequencer to jump to the specified sequence level.

The terms used by the branch qualifier (A through J) are defined by the TERM command. The meaning of **IN_RANGE** and **OUT_RANGE** is determined by the RANGE command.

Within the limitations shown by the syntax definitions, complex expressions may be formed using the **AND** and **OR** operators. Expressions are limited to what you could manually enter through the State Trigger menu. Regarding parentheses, the syntax definitions on the next page show only the required ones. Additional parentheses are allowed as long as the meaning of the expression is not changed. Figure 16-2 shows a complex expression as seen in the State Trigger menu.

Example

The following statements are all correct and have the same meaning. Notice that the conventional rules for precedence are not followed. The expressions are evaluated from left to right.

```
OUTPUT XXX;":MACHINE1:STRIGGER:BRANCH1 'C AND D OR F OR G', 1"
OUTPUT XXX;":MACHINE1:STRIGGER:BRANCH1 '((C AND D) OR
(F OR G))', 1"
OUTPUT XXX;":MACHINE1:STRIGGER:BRANCH1 'F OR (C AND D) OR G',1"
```

`<N>` An integer from 1 to `<number_of_levels>`

`<to_level_number>` An integer from 1 to `<number_of_levels>`

`<number_of_levels>` An integer from 2 to the number of existing sequence levels (maximum 12)

`<branch_qualifier>` `<qualifier>` see "Qualifier" on page 16-7

Examples

```
OUTPUT XXX;":MACHINE1:STRIGGER:BRANCH1 'ANYSTATE', 3"
OUTPUT XXX;":MACHINE2:STRIGGER:BRANCH2 'A', 7"
OUTPUT XXX;":MACHINE1:STRIGGER:BRANCH3 '((A OR B) OR NOTG)', 1"
```

Query

```
:MACHine{1|2}:STRigger:BRANch<N>?
```

The BRANch query returns the current branch qualifier specification for a given sequence level.

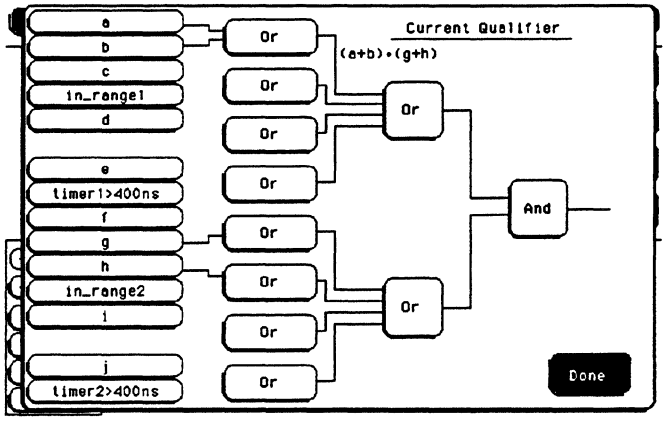
Returned Format

```
[:MACHine{1|2}:STRigger:BRANch<N>] <branch_qualifier>,
<to_level_num><NL>
```

Example

```
OUTPUT XXX;":MACHINE1:STRIGGER:BRANCH3?"
```

Figure 16-2



Complex qualifier

Figure 16-2 is a front panel representation of the complex qualifier (a OR b) AND (g OR h).

Example

This example would be used to specify this complex qualifier.

```
OUTPUT XXX;":MACHINE1:STRIGGER:BRANCH1 '((A OR B) AND
(G OR H))', 2"
```

> Terms A through E, RANGE 1, and TIMER 1 must be grouped together and terms F through J, RANGE 2, and TIMER 2 must be grouped together. In the first level, terms from one group may not be mixed with terms from the other. For example, the expression ((A OR IN_RANGE2) AND (C OR H)) is not allowed because the term C cannot be specified in the E through J group.

In the first level, the operators you can use are **AND, NAND, OR, NOR, XOR, NXOR**. Either **AND** or **OR** may be used at the second level to join the two groups together. It is acceptable for a group to consist of a single term. Thus, an expression like **(B AND G)** is legal, since the two operands are both simple terms from separate groups.

CLEar

Command

```
:MACHine{1|2}:STRigger:CLEar
{All|SEQuence|RESource}
```

The CLEar command allows you to clear all settings in the State Trigger menu and replace them with the default, clear only the Sequence levels, or clear only the resource term patterns.

Example

```
OUTPUT XXX;":MACHINE1:STRIGGER:CLEAR RESOURCE"
```

FIND

Command

```
:MACHine{1|2}:STRigger:FIND<N>
<proceed_qualifier>,<occurrence>
```

The FIND command defines the proceed qualifier for a given sequence level. The qualifier tells the state analyzer when to proceed to the next sequence level. When this proceed qualifier is matched the specified number of times, the sequencer will proceed to the next sequence level. In the sequence level where the trigger is specified, the FIND command specifies the trigger qualifier (see SEQuence command).

The terms A through J are defined by the TERM command. The meaning of `IN_RANGE` and `OUT_RANGE` is determined by the RANGe command. Expressions are limited to what you could manually enter through the State Trigger menu. Regarding parentheses, the syntax definitions below show only the required ones. Additional parentheses are allowed as long as the meaning of the expression is not changed. See figure 16-2 for a detailed example.

`<N>` An integer from 1 to (number of existing sequence levels −1)

`<occurrence>` An integer from 1 to 1048575

`<proceed_qualifier>` `<qualifier>` see "Qualifier" on page 16-7

Examples

```
OUTPUT XXX;":MACHINE1:STRIGGER:FIND1 'ANYSTATE', 1"
OUTPUT XXX;":MACHINE1:STRIGGER:FIND3 '((NOTA AND NOTB) OR G)',
1"
```

Query

```
:MACHine{1|2}:STRigger:FIND4?
```

The FIND query returns the current proceed qualifier specification for a given sequence level.

Returned Format

```
[:MACHine{1|2}:STRigger:FIND<N>] <proceed_qualifier>,
<occurrence><NL>
```

Example

```
OUTPUT XXX;":MACHINE1:STRIGGER:FIND<N>?"
```

RANGe

Command

```
:MACHine{1|2}:STRigger:RANGE
<label_name>,<start_pattern>,<stop_pattern>
```

The RANGe command allows you to specify a range recognizer term for the specified machine. Since a range can only be defined across one label and, since a label must contain 32 or less bits, the value of the start pattern or stop pattern will be between $(2^{32})-1$ and 0.

> Because a label can only be defined across a maximum of two pods, a range term is only available across a single label; therefore, the end points of the range cannot be split between labels.

When these values are expressed in binary, they represent the bit values for the label at one of the range recognizers' end points. Don't cares are not allowed in the end point pattern specifications.

<label_name> String of up to 6 alphanumeric characters

<start_pattern>
```
"{#B{0|1} . . . |
#Q{0|1|2|3|4|5|6|7} . . . |
#H{0|1|2|3|4|5|6|7|8|9|A|B|C|D|E|F} . . . |
{0|1|2|3|4|5|6|7|8|9} . . . }"
```

<stop_pattern>
```
"{#B{0|1} . . . |
#Q{0|1|2|3|4|5|6|7} . . . |
#H{0|1|2|3|4|5|6|7|8|9|A|B|C|D|E|F} . . . |
{0|1|2|3|4|5|6|7|8|9} . . . }"
```

Examples

```
OUTPUT XXX;":MACHINE1:STRIGGER:RANGE 'DATA', '127', '255' "
OUTPUT XXX;":MACHINE1:STRIGGER:RANGE 'ABC', '#B00001111',
'#HCF' "
```

Query

```
:MACHine{1|2}:STRigger:RANGe?
```

The RANGe query returns the range recognizer end point specifications for the range.

Returned Format

```
[:MACHine{1|2}:STRAce:RANGe] <label_name>,<start_pattern>,
<stop_pattern><NL>
```

Example

```
OUTPUT XXX;":MACHINE1:STRIGGER:RANGE?"
```

SEQuence

Command

```
:MACHine{1|2}:STRigger:SEQuence <number_of_levels>,
<level_of_trigger>
```

The SEQuence command redefines the state analyzer trace sequence. First, it deletes the current trace sequence. Then it inserts the number of levels specified, with default settings, and assigns the trigger to be at a specified sequence level. The number of levels can be between 2 and 12 when the analyzer is armed by the RUN key.

`<number_of_levels>` An integer from 2 to 12

`<level_of_trigger>` An integer from 1 to (number of existing sequence levels −1)

Example

```
OUTPUT XXX;":MACHINE1:STRIGGER:SEQUENCE 4,3"
```

Query

```
:MACHine{1|2}:STRigger:SEQuence?
```

The SEQuence query returns the current sequence specification.

Returned Format

```
[:MACHine{1|2}:STRigger:SEQuence] <number_of_levels>,
<level_of_trigger><NL>
```

`<number_of_levels>` An integer from 2 to 12

`<level_of_trigger>` An integer from 1 to (number of existing sequence levels −1)

Example

```
OUTPUT XXX;":MACHINE1:STRIGGER:SEQUENCE?"
```

STORe

Command

```
:MACHine{1|2}:STRigger:STORe<N> <store_qualifier>
```

The STORe command defines the store qualifier for a given sequence level. Any data matching the STORe qualifier will actually be stored in memory as part of the current trace data. The qualifier may be a single term or a complex expression. The terms A through J are defined by the TERM command. The meaning of `IN_RANGE1` and 2 and `OUT_RANGE1` and 2 is determined by the `RANGe` command.

Expressions are limited to what you could manually enter through the State Trigger menu. Regarding parentheses, the syntax definitions below show only the required ones. Additional parentheses are allowed as long as the meaning of the expression is not changed.

A detailed example is provided in figure 16-2 on page 16-12.

`<N>` An integer from 1 to the number of existing sequence levels (maximum 12)

`<store_qualifier>` `<qualifier>` see "Qualifier" on page 16-7

Examples

```
OUTPUT XXX;":MACHINE1:STRIGGER:STORE1 'ANYSTATE'"
OUTPUT XXX;":MACHINE1:STRIGGER:STORE2 'OUT_RANGE1'"
OUTPUT XXX;":MACHINE1:STRIGGER:STORE3 '(NOTC AND NOTD AND
NOTH)'"
```

Query

```
:MACHine{1|2}:STRigger:STORe<N>?
```

The STORe query returns the current store qualifier specification for a given sequence level <N>.

Returned Format

```
[:MACHine{1|2}:STRigger:STORe<N>] <store_qualifier><NL>
```

Example

```
OUTPUT XXX;":MACHINE1:STRIGGER:STORE4?"
```

TAG

Command

```
:MACHine{1|2}:STRigger:TAG
{OFF|TIME|<state_tag_qualifier>}
```

The TAG command selects the type of count tagging (state or time) to be performed during data acquisition. State tagging is indicated when the parameter is the state tag qualifier, which will be counted in the qualified state mode. The qualifier may be a single term or a complex expression. The terms A through J are defined by the TERM command. The terms **IN_RANGE**1 and 2 and **OUT_RANGE**1 and 2 are defined by the **RANGe** command.

Expressions are limited to what you could manually enter through the State Trigger menu. Regarding parentheses, the syntax definitions below show only the required ones. Additional parentheses are allowed as long as the meaning of the expression is not changed. A detailed example is provided in figure 16-2 on page 16-12.

`<state_tag_qualifier>` `<qualifier>` see "Qualifier" on page 16-7

Examples

```
OUTPUT XXX;":MACHINE1:STRIGGER:TAG OFF"
OUTPUT XXX;":MACHINE1:STRIGGER:TAG TIME"
OUTPUT XXX;":MACHINE1:STRIGGER:TAG '(IN_RANGE OR NOTF)'"
OUTPUT XXX;":MACHINE1:STRIGGER:TAG '((IN_RANGE OR A) AND E)'"
```

Query

```
:MACHine{1|2}:STRigger:TAG?
```

The TAG query returns the current count tag specification.

Returned Format

```
[:MACHine{1|2}:STRigger:TAG]
{OFF|TIME|<state_tag_qualifier>}<NL>
```

Example

```
OUTPUT XXX;":MACHINE1:STRIGGER:TAG?"
```

TAKenbranch

Command

```
:MACHine{1|2}:STRigger:TAKenbranch {STORe|NOSTore}
```

The TAKenbranch command allows you to specify whether the state causing a sequence level change is stored or not stored for the specified machine. Both a state that causes the sequencer to proceed or a state that causes the sequencer to branch is considered a sequence level change. A branch can also jump to itself and this also considered a sequence level change. The state causing the branch is defined by the BRANch command.

Example

```
OUTPUT XXX;":MACHINE2:STRIGGER:TAKENBRANCH STORE"
```

Query

```
:MACHine{1|2}:STRigger:TAKenbranch?
```

The TAKenbranch query returns the current setting.

Returned Format

```
[:MACHine{1|2}:STRigger:TAKenbranch] {STORe|NOSTore}<NL>
```

Example

```
OUTPUT XXX;":MACHINE2:STRIGGER:TAKENBRANCH?
```

TCONtrol

Command

```
:MACHine{1|2}:STRigger:TCONtrol<N> <timer_num>,
{OFF|STARt|PAUSe|CONTinue}
```

The TCONtrol (timer control) command allows you to turn off, start, pause, or continue the timer for the specified level. The time value of the timer is defined by the TIMER command. There are two timers and they are independently available for either machine. Neither timer can be assigned to both machines simultaneously.

`<N>` An integer from 1 to the number of existing sequence levels (maximum 12)

`<timer_num>` `{1|2}`

Example

```
OUTPUT XXX;":MACHINE2:STRIGGER:TCONTROL6 1, PAUSE"
```

Query

```
:MACHine{1|2}:STRigger:TCONTROL<N>? <timer_num>
```

The TCONtrol query returns the current TCONtrol setting of the specified level.

Returned Format

```
[:MACHine{1|2}:STRigger:TCONTROL<N> <timer_num>]
{OFF|STARt|PAUSe|CONTinue}<NL>
```

`<N>` An integer from 1 to the number of existing sequence levels (maximum 12)

`<timer_num>` `{1|2}`

Example

```
OUTPUT XXX;":MACHINE2:STRIGGER:TCONTROL?6 1"
```

TERM

Command

```
:MACHine{1|2}:STRigger:TERM <term_id>,<label_name>,
<pattern>
```

The TERM command allows you to specify a pattern recognizer term in the specified machine. Each command deals with only one label in the given term; therefore, a complete specification could require several commands. Since a label can contain 32 or less bits, the range of the pattern value will be between $2^{32} - 1$ and 0. When the value of a pattern is expressed in binary, it represents the bit values for the label inside the pattern recognizer term. Because the pattern parameter may contain don't cares and be represented in several bases, it is handled as a string of characters rather than a number.

All 10 terms (A through J) are available for either machine but not both simultaneously. If you send the TERM command to a machine with a term that has not been assigned to that machine, an error message "Legal command but settings conflict" is returned.

`<term_id>` `{A|B|C|D|E|F|G|H|I|J}`

`<label_name>` A string of up to 6 alphanumeric characters

`<pattern>`

```
"{#B{0|1|X} . . . |
#Q{0|1|2|3|4|5|6|7|X} . . . |
#H{0|1|2|3|4|5|6|7|8|9|A|B|C|D|E|F|X} . . . |
{0|1|2|3|4|5|6|7|8|9} . . . }"
```

Example

```
OUTPUT XXX;":MACHINE1:STRIGGER:TERM A,'DATA','255' "
OUTPUT XXX;":MACHINE1:STRIGGER:TERM B,'ABC','#BXXXX1101' "
```

Query

```
:MACHine{1|2}:STRigger:TERM? <term_id>,<label_name>
```

The TERM query returns the specification of the term specified by term identification and label name.

Returned Format

```
[:MACHine{1|2}:STRAce:TERM]
<term_id>,<label_name>,<pattern><NL>
```

Example

```
OUTPUT XXX;":MACHINE1:STRIGGER:TERM? B,'DATA' "
```

TIMER

Command

```
:MACHine{1|2}:STRigger:TIMER{1|2} <time_value>
```

The TIMER command sets the time value for the specified timer. The limits of the timer are 400 ns to 500 seconds in 16 ns to 500 μs increments. The increment value varies with the time value of the specified timer. There are two timers and they are independently available for either machine. Neither timer can be assigned to both machines simultaneously.

`<time_value>` A real number from 400 ns to 500 seconds in increments which vary from 16 ns to 500 μs.

Example

```
OUTPUT XXX;":MACHINE1:STRIGGER:TIMER1 100E-6"
```

Query `:MACHine{1|2}:STRigger:TIMER{1|2}?`

The TIMER query returns the current time value for the specified timer.

Returned Format `[:MACHine{1|2}:STRigger:TIMER{1|2}] <time_value><NL>`

`<time_value>` A real number from 400 ns to 500 seconds in increments which vary from 16 ns to 500 μs.

Example

```
OUTPUT XXX;":MACHINE1:STRIGGER:TIMER1?"
```

TPOSition

Command `:MACHine{1|2}:STRigger:TPOSition {STARt|CENTer|END|POSTstore,<poststore>}`

The TPOSition (trigger position) command allows you to set the trigger at the start, center, end or at any position in the trace (poststore). When STARt is specified, approximately 16 states are stored before the trigger. When END is specified, approximately 16 states are stored after the trigger. Poststore is defined as 0 to 100 percent. When 0 or 100 percent is specified, the trigger is actually the first or last state respectively.

`<poststore>` An integer from 0 to 100 representing percentage of poststore.

Examples

```
OUTPUT XXX;":MACHINE1:STRIGGER:TPOSITION END"
OUTPUT XXX;":MACHINE1:STRIGGER:TPOSITION POSTstore,75"
```

Query

```
:MACHine{1|2}:STRigger:TPOSition?
```

The TPOSition query returns the current trigger position setting.

Returned Format

```
[:MACHine{1|2}:STRigger:TPOSition] {STARt|CENTer|END|
POSTstore,<poststore>}<NL>
```

Example

```
OUTPUT XXX;":MACHINE1:STRIGGER:TPOSITION?"
```

17

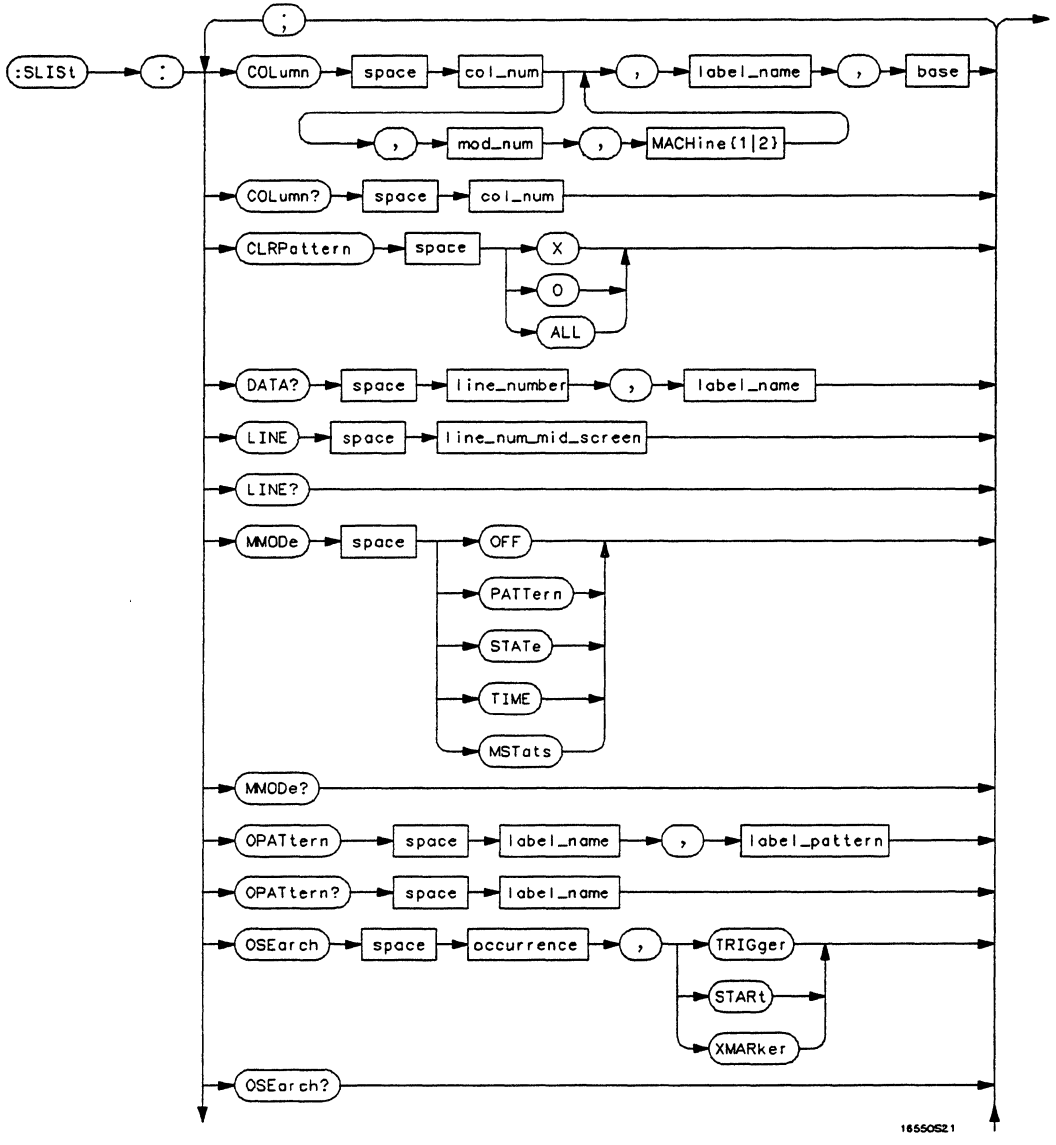
SLISt Subsystem

Introduction

The SLISt subsystem contains the commands available for the State Listing menu in the HP 1660A logic analyzer. These commands are:

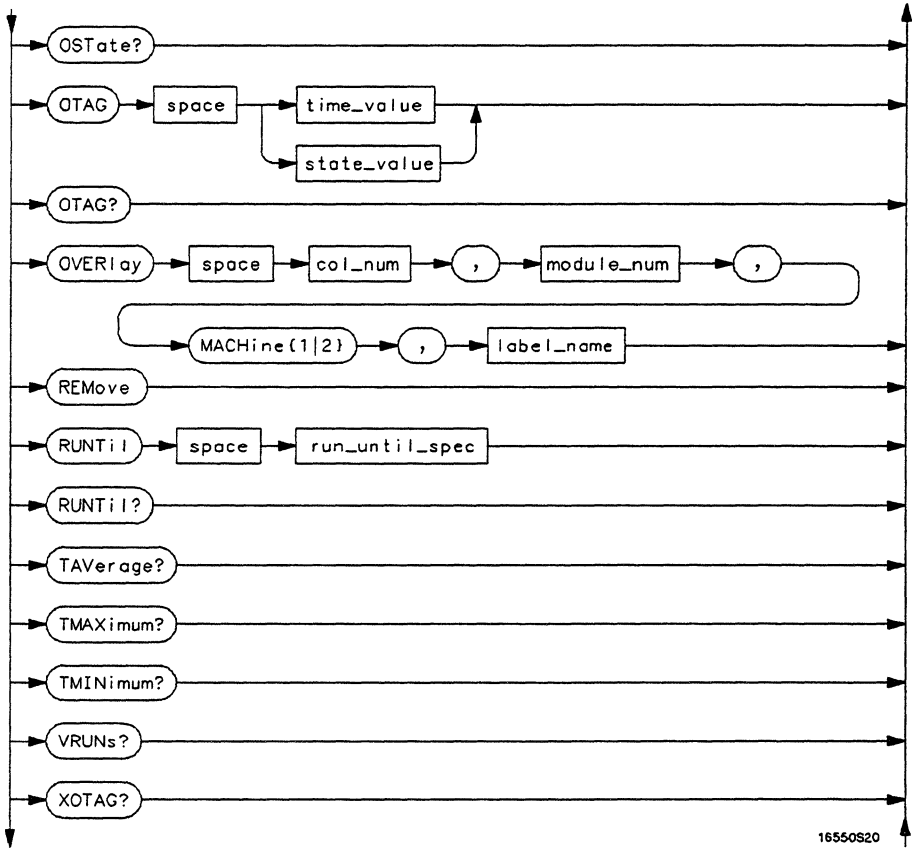
- COLumn
- CLRPattern
- DATA
- LINE
- MMODe
- OPATtern
- OSEarch
- OSTate
- OTAG
- OVERlay
- REMove
- RUNTil
- TAVerage
- TMAXimum
- TMINimum
- VRUNs
- XOTag
- XOTime
- XPATtern
- XSEarch
- XSTate
- XTAG

Figure 17-1



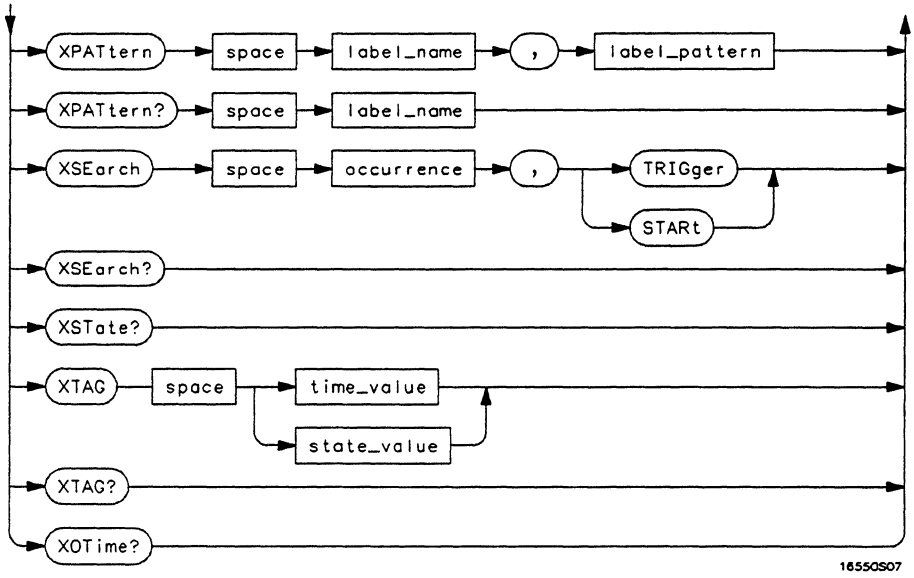
SLISt Subsystem Syntax Diagram

Figure 17-1 (continued)



SLISt Subsystem Syntax Diagram (continued)

Figure 17-1 (continued)



SLISt Subsystem Syntax Diagram (continued)

Table 7-1 SLISt Parameter Values

Parameter	Values
module_num	{1|2|3|4|5|6|7|8} (2 through 10 not used)
mach_num	{1|2}
col_num	Integer from 1 to 61
line_number	Integer from −8191 to +8191
label_name	A string of up to 6 alphanumeric characters
base	{BINary|HEXadecimal|OCTal|DECimal|TWOS|ASCii|SYMBol|IASSembler} for labels or {ABSolute|RELative} for tags
line_num_mid_screen	Integer from −8191 to +8191
label_pattern	"{#B{0|1|X}...| #Q{0|1|2|3|4|5|6|7|X}...| #H{0|1|2|3|4|5|6|7|8|9|A|B|C|D|E|F|X}...| {0|1|2|3|4|5|6|7|8|9}...}"
occurrence	Integer from −8191 to +8192
time_value	Real number
state_value	Real number
run_until_spec	{OFF|LT,<value>|GT,<value>|INRange,<value>,<value>|OUTRange,<value>,<value>}
value	Real number

SLISt

Selector

```
:MACHine{1|2}:SLISt
```

The SLISt selector is used as part of a compound header to access those settings normally found in the State Listing menu. It always follows the MACHine selector because it selects a branch directly below the MACHine level in the command tree.

Example

```
OUTPUT XXX;":MACHINE1:SLIST:LINE 256"
```

COLumn

Command

```
:MACHine{1|2}:SLISt:COLumn <col_num>
[,<module_num>, MACHine{1|2}],<label_name>,<base>
```

The COLumn command allows you to configure the state analyzer list display by assigning a label name and base to one of the 61 vertical columns in the menu. A column number of 1 refers to the left most column. When a label is assigned to a column it replaces the original label in that column.

When the label name is "TAGS," the TAGS column is assumed and the next parameter must specify RELative or ABSolute.

> A label for tags must be assigned in order to use ABSolute or RELative state tagging.

<col_num>	integer from 1 to 61
<module_num>	{1|2|3|4|5|6|7|8|9|10} (2 through 10 not used)
<label_name>	string of up to 6 alphanumeric characters
<base>	{BINary|HEXadecimal|OCTal|DECimal|TWOS|ASCii|SYMBol|IASSembler} for labels or {ABSolute|RELative} for tags

Example

```
OUTPUT XXX;":MACHINE1:SLIST:COLUMN 4,'A',HEX"
```

Query

```
:MACHine{1|2}:SLISt:COLumn? <col_num>
```

The COLumn query returns the column number, label name, and base for the specified column.

Returned Format

```
[:MACHine{1|2}:SLISt:COLumn] <col_num>,<module_num>,
MACHine{1|2},<label_name>,<base><NL>
```

Example

```
OUTPUT XXX;":MACHINE1:SLIST:COLUMN? 4"
```

CLRPattern

Command

```
:MACHine{1|2}:SWAVeform:CLRPattern {X|O|ALL}
```

The CLRPattern command allows you to clear the patterns in the selected Specify Patterns menu.

Example

```
OUTPUT XXX;":MACHINE1:SWAVEFORM:CLRPATTERN X"
```

DATA

Query

```
:MACHine{1|2}:SLISt:DATA?
<line_number>,<label_name>
```

The DATA query returns the value at a specified line number for a given label. The format will be the same as the one shown in the listing display.

Returned Format

```
[:MACHine{1|2}:SLISt:DATA] <line_number>,<label_name>,
<pattern_string><NL>
```

<line_number> integer from −8191 to +8191

<label_name> string of up to 6 alphanumeric characters

<pattern_string>

```
"{#B{0|1|X} . . . |
#Q{0|1|2|3|4|5|6|7|X} . . . |
#H{0|1|2|3|4|5|6|7|8|9|A|B|C|D|E|F|X} . . . |
{0|1|2|3|4|5|6|7|8|9} . . . }"
```

Example

```
OUTPUT XXX;":MACHINE1:SLIST:DATA? 512, 'RAS'"
```

LINE

Command

```
:MACHine{1|2}:SLISt:LINE <line_num_mid_screen>
```

The LINE command allows you to scroll the state analyzer listing vertically. The command specifies the state line number relative to the trigger that the analyzer highlights at the center of the screen.

<line_num_mid_screen> integer from −8191 to +8191

Example

```
OUTPUT XXX;":MACHINE1:SLIST:LINE 0"
```

Query

```
:MACHine{1|2}:SLISt:LINE?
```

The LINE query returns the line number for the state currently in the box at the center of the screen.

Returned Format

```
[:MACHine{1|2}:SLISt:LINE] <line_num_mid_screen><NL>
```

Example

```
OUTPUT XXX;":MACHINE1:SLIST:LINE?"
```

MMODe

Command

```
:MACHine{1|2}:SLISt:MMODe <marker_mode>
```

The MMODe command (Marker Mode) selects the mode controlling the marker movement and the display of marker readouts. When PATTern is selected, the markers will be placed on patterns. When STATe is selected and state tagging is on, the markers move on qualified states counted between normally stored states. When TIME is selected and time tagging is enabled, the markers move on time between stored states. When MSTats is selected and time tagging is on, the markers are placed on patterns, but the readouts will be time statistics.

<marker_mode>

```
{OFF|PATTern|STATe|TIME|MSTats}
```

Example

```
OUTPUT XXX;":MACHINE1:SLIST:MMODE TIME"
```

Query

```
:MACHine{1|2}:SLISt:MMODe?
```

The MMODe query returns the current marker mode selected.

Returned Format:

```
[:MACHine{1|2}:SLISt:MMODe] <marker_mode><NL>
```

Example

```
OUTPUT XXX;":MACHINE1:SLIST:MMODE?"
```

OPATtern

Command

```
:MACHine{1|2}:SLISt:OPATtern
<label_name>,<label_pattern>
```

The OPATtern command allows you to construct a pattern recognizer term for the O Marker which is then used with the OSEarch criteria when moving the marker on patterns. Because this command deals with only one label at a time, a complete specification could require several invocations.

When the value of a pattern is expressed in binary, it represents the bit values for the label inside the pattern recognizer term. In whatever base is used, the value must be between 0 and $2^{32} - 1$, since a label may not have more than 32 bits. Because the `<label_pattern>` parameter may contain don't cares, it is handled as a string of characters rather than a number.

`<label_name>` string of up to 6 alphanumeric characters

`<label_pattern>`

```
"{#B{0|1|X} . . . |
#Q{0|1|2|3|4|5|6|7|X} . . . |
#H{0|1|2|3|4|5|6|7|8|9|A|B|C|D|E|F|X} . . . |
{0|1|2|3|4|5|6|7|8|9} . . . }"
```

Examples

```
OUTPUT XXX;":MACHINE1:SLIST:OPATTERN 'DATA','255' "
OUTPUT XXX;":MACHINE1:SLIST:OPATTERN 'ABC','#BXXXX1101' "
```

Query

```
:MACHine{1|2}:SLISt:OPATtern? <label_name>
```

The OPATtern query returns the pattern specification for a given label name.

Returned Format

```
[:MACHine{1|2}:SLISt:OPATtern] <label_name>,<label_pattern><NL>
```

Example

```
OUTPUT XXX;":MACHINE1:SLIST:OPATTERN? 'A'"
```

OSEarch

Command

```
:MACHine{1|2}:SLISt:OSEarch <occurrence>,<origin>
```

The OSEarch command defines the search criteria for the O marker, which is then used with associated OPATtern recognizer specification when moving the markers on patterns. The origin parameter tells the marker to begin a search with the trigger, the start of data, or with the X marker. The actual occurrence the marker searches for is determined by the occurrence parameter of the OSEarch recognizer specification, relative to the origin. An occurrence of 0 places the marker on the selected origin. With a negative occurrence, the marker searches before the origin. With a positive occurrence, the marker searches after the origin.

`<occurrence>` integer from −8191 to +8191

`<origin>` `{TRIGger|STARt|XMARker}`

Example

```
OUTPUT XXX;":MACHINE1:SLIST:OSEARCH +10,TRIGGER"
```

Query

```
:MACHine{1|2}:SLISt:OSEarch?
```

The OSEarch query returns the search criteria for the O marker.

Returned Format

```
[:MACHine{1|2}:SLISt:OSEarch] <occurrence>,<origin><NL>
```

Example

```
OUTPUT XXX;":MACHINE1:SLIST:OSEARCH?"
```

OSTate

Query

```
:MACHine{1|2}:SLISt:OSTate?
```

The OSTate query returns the line number in the listing where the O marker resides (−8191 to +8191). If data is not valid, the query returns 32767.

Returned Format

```
[:MACHine{1|2}:SLISt:OSTate] <state_num><NL>
```

`<state_num>` an integer from −8191 to +8191, or 32767

Example

```
OUTPUT XXX;":MACHINE1:SLIST:OSTATE?"
```

OTAG

Command

```
:MACHine{1|2}:SLISt:OTAG
<time_value>|<state_value>}
```

The OTAG command specifies the tag value on which the O Marker should be placed. The tag value is time when time tagging is on, or states when state tagging is on. If the data is not valid tagged data, no action is performed.

`<time_value>` real number

`<state_value>` integer

Example

```
:OUTPUT XXX;":MACHINE1:SLIST:OTAG 40.0E−6"
```

Query

```
:MACHine{1|2}:SLISt:OTAG?
```

The OTAG query returns the O Marker position in time when time tagging is on or in states when state tagging is on, regardless of whether the marker was positioned in time or through a pattern search. If data is not valid, the query returns 9.9E37 for time tagging, or returns 32767 for state tagging.

Returned Format

```
[:MACHine{1|2}:SLISt:OTAG] {time_value>|<state_value>}<NL>
```

`<time_value>` real number

`<state_value>` integer

Example

```
OUTPUT XXX;":MACHINE1:SLIST:OTAG?"
```

OVERlay

Command

```
:MACHine{1|2}:SLISt:OVERlay <col_num>,<module_num>,
MACHine{1|2},<label_name>
```

The OVERlay command allows you to add time-correlated labels from other modules or machines to the state listing.

`<col_num>` integer from 1 to 61

`<Module_num>` integer 1 through 10 (2 through 10 unused)

`<label_name>` string of up to 6 alphanumeric characters

Example

```
OUTPUT XXX;":MACHINE1:SLIST:OVERlay,25,5,MACHINE2,'DATA'"
```

REMove

Command

```
:MACHine{1|2}:SLISt:REMove
```

The REMove command removes all labels, except the leftmost label, from the listing menu.

Example

```
OUTPUT XXX;":MACHINE1:SLIST:REMOVE"
```

RUNTil

Command

```
:MACHine{1|2}:SLISt:RUNTil <run_until_spec>
```

The RUNTil (run until) command allows you to define a stop condition when the trace mode is repetitive. Specifying OFF causes the analyzer to make runs until either the display's STOP field is touched, or, when the STOP command is issued.

There are four conditions based on the time between the X and O markers. Using this difference in the condition is effective only when time tags have been turned on (see the TAG command in the STRace subsystem). These four conditions are as follows:

- The difference is less than (LT) some value.
- The difference is greater than (GT) some value.
- The difference is inside some range (INRange).
- The difference is outside some range (OUTRange).

End points for the INRange and OUTRange should be at least 8 ns apart since this is the minimum time resolution of the time tag counter.

There are two conditions which are based on a comparison of the acquired state data and the compare data image. The analyzer can run until one of the following conditions is true:

- Every channel of every label has the same value (EQUal).
- Any channel of any label has a different value (NEQual).

> The RUNTil instruction (for state analysis) is available in both the SLISt and COMPare subsystems.

`<run_until_spec>` `{OFF|LT,<value>|GT,<value>|INRange,<value>,<value>|OUTRange,<value>,<value>|EQUal|NEQual}`

`<value>` real number from −9E9 to +9E9

Example

```
OUTPUT XXX;":MACHINE1:SLIST:RUNTIL GT,800.0E-6"
```

Query

```
:MACHine{1|2}:SLISt:RUNTil?
```

The RUNTil query returns the current stop criteria.

Returned Format

```
[:MACHine{1|2}:SLISt:RUNTil] <run_until_spec><NL>
```

Example

```
OUTPUT XXX;":MACHINE1:SLIST:RUNTIL?"
```

TAVerage

Query `:MACHine{1|2}:SLISt:TAVerage?`

The TAVerage query returns the value of the average time between the X and O Markers. If the number of valid runs is zero, the query returns 9.9E37. Valid runs are those where the pattern search for both the X and O markers was successful, resulting in valid delta-time measurements.

Returned Format: `[:MACHine{1|2}:SLISt:TAVerage] <time_value><NL>`

`<time_value>` real number

Example

```
OUTPUT XXX;":MACHINE1:SLIST:TAVERAGE?"
```

TMAXimum

Query `:MACHine{1|2}:SLISt:TMAXimum?`

The TMAXimum query returns the value of the maximum time between the X and O Markers. If data is not valid, the query returns 9.9E37.

Returned Format `[:MACHine{1|2}:SLISt:TMAXimum] <time_value><NL>`

`<time_value>` real number

Example

```
OUTPUT XXX;":MACHINE1:SLIST:TMAXIMUM?"
```

TMINimum

Query `:MACHine{1|2}:SLISt:TMINimum?`

The TMINimum query returns the value of the minimum time between the X and O Markers. If data is not valid, the query returns 9.9E37.

Returned Format `[:MACHine{1|2}:SLISt:TMINimum] <time_value><NL>`

`<time_value>` real number

Example:

```
OUTPUT XXX;":MACHINE1:SLIST:TMINIMUM?"
```

VRUNs

Query `:MACHine{1|2}:SLISt:VRUNs?`

The VRUNs query returns the number of valid runs and total number of runs made. Valid runs are those where the pattern search for both the X and O markers was successful resulting in valid delta time measurements.

Returned Format `[:MACHine{1|2}:SLISt:VRUNs] <valid_runs>,<total_runs><NL>`

`<valid_runs>` zero or positive integer

`<total_runs>` zero or positive integer

Example:

```
OUTPUT XXX;":MACHINE1:SLIST:VRUNS?"
```

XOTag

Query `:MACHine{1|2}:SLISt:XOTag?`

The XOTag query returns the time from the X to O markers when the marker mode is time or number of states from the X to O markers when the marker mode is state. If there is no data in the time mode the query returns 9.9E37. If there is no data in the state mode, the query returns 32767.

Returned Format `[:MACHine{1|2}:SLISt:XOTag] {<XO_time>|<XO_states>}<NL>`

`<XO_time>` real number

`<XO_states>` integer

Example

```
OUTPUT XXX;":MACHINE1:SLIST:XOTAG?"
```

XOTime

Query `:MACHine{1|2}:SLISt:XOTime?`

The XOTime query returns the time from the X to O markers when the marker mode is time or number of states from the X to O markers when the marker mode is state. If there is no data in the time mode the query returns 9.9E37. If there is no data in the state mode, the query returns 32767.

Returned Format `[:MACHine{1|2}:SLISt:XOTime] {<XO_time>|<XO_states>}<NL>`

`<XO_time>` real number

`<XO_states>` integer

Example

```
OUTPUT XXX;":MACHINE1:SLIST:XOTIME?"
```

XPATtern

Command

```
:MACHine{1|2}:SLISt:XPATtern <label_name>,
<label_pattern>
```

The XPATtern command allows you to construct a pattern recognizer term for the X Marker which is then used with the XSEarch criteria when moving the marker on patterns. Since this command deals with only one label at a time, a complete specification could require several invocations.

When the value of a pattern is expressed in binary, it represents the bit values for the label inside the pattern recognizer term. In whatever base is used, the value must be between 0 and $2^{32} - 1$, since a label may not have more than 32 bits. Because the `<label_pattern>` parameter may contain don't cares, it is handled as a string of characters rather than a number.

`<label_name>` string of up to 6 alphanumeric characters

`<label_pattern>`

```
"{#B{0|1|X} . . . |
#Q{0|1|2|3|4|5|6|7|X} . . . |
#H{0|1|2|3|4|5|6|7|8|9|A|B|C|D|E|F|X} . . . |
{0|1|2|3|4|5|6|7|8|9} . . . }"
```

Examples

```
OUTPUT XXX;":MACHINE1:SLIST:XPATTERN 'DATA','255' "
OUTPUT XXX;":MACHINE1:SLIST:XPATTERN 'ABC','#BXXXX1101' "
```

Query

```
:MACHine{1|2}:SLISt:XPATtern? <label_name>
```

The XPATtern query returns the pattern specification for a given label name.

Returned Format

```
[:MACHine{1|2}:SLISt:XPATtern] <label_name>,<label_pattern><NL>
```

Example

```
OUTPUT XXX;":MACHINE1:SLIST:XPATTERN? 'A'"
```

XSEarch

Command

```
:MACHine{1|2}:SLISt:XSEarch <occurrence>,<origin>
```

The XSEarch command defines the search criteria for the X Marker, which is then with associated XPATtern recognizer specification when moving the markers on patterns. The origin parameter tells the Marker to begin a search with the trigger or with the start of data. The occurrence parameter determines which occurrence of the XPATtern recognizer specification, relative to the origin, the marker actually searches for. An occurrence of 0 places a marker on the selected origin.

`<occurrence>` integer from −8191 to +8191

`<origin>` `{TRIGger|STARt}`

Example

```
OUTPUT XXX;":MACHINE1:SLIST:XSEARCH +10,TRIGGER"
```

Query

```
:MACHine{1|2}:SLISt:XSEarch?
```

The XSEarch query returns the search criteria for the X marker.

Returned Format

```
[:MACHine{1|2}:SLISt:XSEarch] <occurrence>,<origin><NL>
```

Example

```
OUTPUT XXX;":MACHINE1:SLIST:XSEARCH?"
```

XSTate

Query

```
:MACHine{1|2}:SLISt:XSTate?
```

The XSTate query returns the line number in the listing where the X marker resides (−8191 to +8191). If data is not valid, the query returns 32767.

Returned Format

```
[:MACHine{1|2}:SLISt:XSTate] <state_num><NL>
```

`<state_num>` integer from −8191 to +8191, or 32767

Example

```
OUTPUT XXX;":MACHINE1:SLIST:XSTATE?"
```

XTAG

Command

```
:MACHine{1|2}:SLISt:XTAG
{<time_value>|<state_value>}
```

The XTAG command specifies the tag value on which the X Marker should be placed. The tag value is time when time tagging is on or states when state tagging is on. If the data is not valid tagged data, no action is performed.

`<time_value>` real number

`<state_value>` integer

Example

```
OUTPUT XXX;":MACHINE1:SLIST:XTAG 40.0E-6"
```

Query `:MACHine{1|2}:SLISt:XTAG?`

The XTAG query returns the X Marker position in time when time tagging is on or in states when state tagging is on, regardless of whether the marker was positioned in time or through a pattern search. If data is not valid tagged data, the query returns 9.9E37 for time tagging, or retruns 32767 for state tagging.

Returned Format `[:MACHine{1|2}:SLISt:XTAG] {<time_value>|<state_value>}<NL>`

Example

```
OUTPUT XXX;":MACHINE1:SLIST:XTAG?"
```

18

SWAVeform Subsystem

Introduction

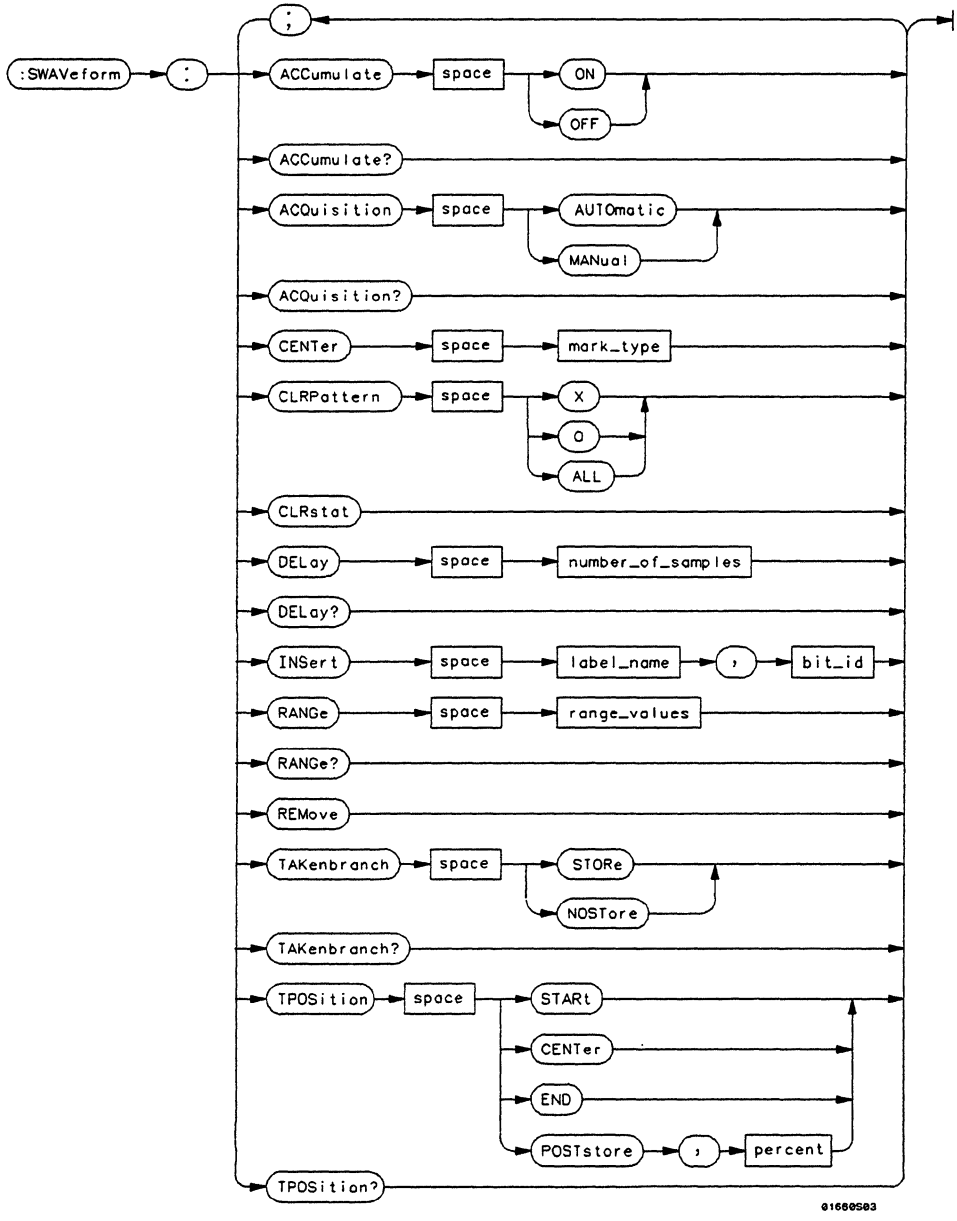
The commands in the State Waveform subsystem allow you to configure the display so that you can view state data as waveforms on up to 96 channels identified by label name and bit number. The 11 commands are analogous to their counterparts in the Timing Waveform subsystem. However, in this subsystem the x-axis is restricted to representing only samples (states), regardless of whether time tagging is on or off. As a result, the only commands which can be used for scaling are DELay and RANge.

The way to manipulate the X and O markers on the Waveform display is through the State Listing (SLISt) subsystem. Using the marker commands from the SLISt subsystem will affect the markers on the Waveform display.

The commands in the SWAVeform subsystem are:

- ACCumulate
- ACQuisition
- CENter
- CLRPattern
- CLRStat
- DELay
- INSert
- RANGe
- REMove
- TAKenbranch
- TPOSition

Figure 18-1



SWAVeform Subsystem Syntax Diagram

Table 18-1 **SWAVeform Parameter Values**

Parameter	Value
number_of_samples	integer from −8191 to +8191
label_name	string of up to 6 alphanumeric characters
bit_id	{OVERlay|<bit_num>|ALL}
bit_num	integer representing a label bit from 0 to 31
range_values	integer from 10 to 5000 (representing (10 × states/Division))
mark_type	{X|O|XO|TRIGger}
percent	integer from 0 to 100

SWAVeform

Selector

```
:MACHine{1|2}:SWAVeform
```

The SWAVeform (State Waveform) selector is used as part of a compound header to access the settings in the State Waveform menu. It always follows the MACHine selector because it selects a branch directly below the MACHine level in the command tree.

Example

```
OUTPUT XXX;":MACHINE2:SWAVEFORM:RANGE 40"
```

ACCumulate

Command

```
:MACHine{1|2}:SWAVeform:ACCumulate {{ON|1}|{OFF|0}}
```

The ACCumulate command allows you to control whether the waveform display gets erased between individual runs or whether subsequent waveforms are allowed to be displayed over the previous waveforms.

Example

```
OUTPUT XXX;":MACHINE1:SWAVEFORM:ACCUMULATE ON"
```

Query

```
:MACHine{1|2}:SWAVeform:ACCumulate?
```

The ACCumulate query returns the current setting. The query always shows the setting as the characters, "0" (off) or "1" (on).

Returned Format

```
[:MACHine{1|2}:SWAVeform:ACCumulate] {0|1}<NL>
```

Example

```
OUTPUT XXX;":MACHINE1:SWAVEFORM:ACCUMULATE?"
```

ACQuisition

Command

```
:MACHine{1|2}:SWAVeform:ACQuisition
{AUTOmatic|MANual}
```

The ACQuisition command allows you to specify the acquisition mode for the state analyzer. The acquisition modes are automatic and manual.

Example

```
OUTPUT XXX;":MACHINE2:SWAVEFORM:ACQUISITION AUTOMATIC"
```

Query `:MACHine{1|2}:SWAVeform:ACQuisition?`

The ACQusition query returns the current acquisition mode.

Returned Format `[:MACHine{1|2}:SWAVeform:ACQuisition] {AUTOmatic|MANual}<NL>`

Example

```
OUTPUT XXX;":MACHINE2:SWAVEFORM:ACQUISITION?"
```

CENTer

Command `:MACHine{1|2}:SWAVeform:CENTer <marker_type>`

The CENTer command allows you to center the waveform display about the specified markers. The markers are placed on the waveform in the SLISt subsystem.

`<marker_type>` `{X|O|XO|TRIGger}`

Example

```
OUTPUT XXX;":MACHINE1:SWAVEFORM:CENTER X"
```

CLRPattern

Command `:MACHine{1|2}:SWAVeform:CLRPattern {X|O|ALL}`

The CLRPattern command allows you to clear the patterns in the selected Specify Patterns menu.

Example

```
OUTPUT XXX;":MACHINE1:SWAVEFORM:CLRPATTERN"
```

CLRStat

Command

```
:MACHine{1|2}:SWAVeform:CLRStat
```

The CLRStat command allows you to clear the waveform statistics without having to stop and restart the acquisition.

Example

```
OUTPUT XXX;":MACHINE1:SWAVEFORM:CLRSTAT"
```

DELay

Command

```
:MACHine{1|2}:SWAVeform:DELay <number_of_samples>
```

The DELay command allows you to specify the number of samples between the State trigger and the horizontal center of the screen for the waveform display. The allowed number of samples is from −8191 to +8191.

`<number_of_samples>` integer from −8191 to +8191

Example

```
OUTPUT XXX;":MACHINE2:SWAVEFORM:DELAY 127"
```

Query

```
:MACHine{1|2}:SWAVeform:DELay?
```

The DELay query returns the current sample offset value.

Returned Format

```
[:MACHine{1|2}:SWAVeform:DELay] <number_of_samples><NL>
```

`<number_of_samples>` integer from −8191 to +8191

Example

```
OUTPUT XXX;":MACHINE1:SWAVEFORM:DELAY?"
```

INSert

Command

```
:MACHine{1|2}:SWAVeform:INSert
<label_name>,<bit_id>
```

The INSert command allows you to add waveforms to the state waveform display. Waveforms are added from top to bottom on the screen. When 96 waveforms are present, inserting additional waveforms replaces the last waveform. Bit numbers are zero based, so a label with 8 bits is referenced as bits 0 through 7. Specifying OVERlay causes a composite waveform display of all bits or channels for the specified label.

`<label_name>` string of up to 6 alphanumeric characters

`<bit_id>` `{OVERlay|<bit_num>ALL}`

`<bit_num>` integer representing a label bit from 0 to 31

Examples

```
OUTPUT XXX;":MACHINE1:SWAVEFORM:INSERT 'WAVE', 19"
OUTPUT XXX;":MACHINE1:SWAVEFORM:INSERT 'ABC', OVERLAY"
OUTPUT XXX;":MACH1:SWAV:INSERT 'POD1', #B1001"
```

RANGe

Command

```
:MACHine{1|2}:SWAVeform:RANGe <number_of_samples>
```

The RANGe command allows you to specify the number of samples across the screen on the State Waveform display. It is equivalent to ten times the states per division setting (states/Div) on the front panel. A number between 10 and 5000 may be entered.

`<number_of_samples>` integer from 10 to 5000

Example

```
OUTPUT XXX;":MACHINE2:SWAVEFORM:RANGE 80"
```

Query

```
:MACHine{1|2}:SWAVeform:RANGe?
```

The RANGe query returns the current range value.

Returned Format

```
[:MACHine{1|2}:SWAVeform:RANGe] <number_of_samples><NL>
```

`<number_of_samples>` integer from 10 to 5000

Example

```
OUTPUT XXX;":MACHINE2:SWAVEFORM:RANGE?"
```

REMove

Command

```
:MACHine{1|2}:SWAVeform:REMove
```

The REMove command allows you to clear the waveform display before building a new display.

Example

```
OUTPUT XXX;":MACHINE1:SWAVEFORM:REMOVE"
```

TAKenbranch

Command

```
:MACHine{1|2}:SWAVeform:TAKenbranch {STORe|NOSTore}
```

The TAKenbranch command allows you to control whether the states that cause branching are stored or not stored. This command is only available when the acquisition mode is set to manual.

Example

```
OUTPUT XXX;":MACHINE2:SWAVEFORM:TAKENBRANCH STORE"
```

Query

```
:MACHine{1|2}:SWAVeform:TAKenbranch?
```

The TAKenbranch query returns the current setting.

Returned Format

```
[:MACHine{1|2}:SWAVeform:TAKenbranch] {STORe|NOSTore}<NL>
```

Example

```
OUTPUT XXX;":MACHINE2:SWAVEFORM:TAKENBRANCH?"
```

TPOSition

Command

```
:MACHine{1|2}:SWAVeform:TPOSition
{STARt|CENTer|END|POSTstore,<percent>}
```

The TPOSition command allows you to control where the trigger point is placed. The trigger point can be placed at the start, center, end, or at a percentage of post store. The post store option is the same as the User Defined option when setting the trigger point from the front panel.

The TPOSition command is only available when the acquisition mode is set to manual.

`<percent>` integer from 1 to 100

Example

```
OUTPUT XXX;":MACHINE2:SWAVEFORM:TPOSITION CENTER"
```

Query `:MACHine{1|2}:SWAVeform:TPOSition?`

The TPOSition query returns the current trigger setting.

Returned Format

```
[:MACHine{1|2}:SWAVeform:TPOSition]
{STARt|CENTer|END|POSTstore,
<percent>}<NL>
```

`<percent>` integer from 1 to 100

Example

```
OUTPUT XXX;":MACHINE2:SWAVEFORM:TPOSition?"
```

19

SCHart Subsystem

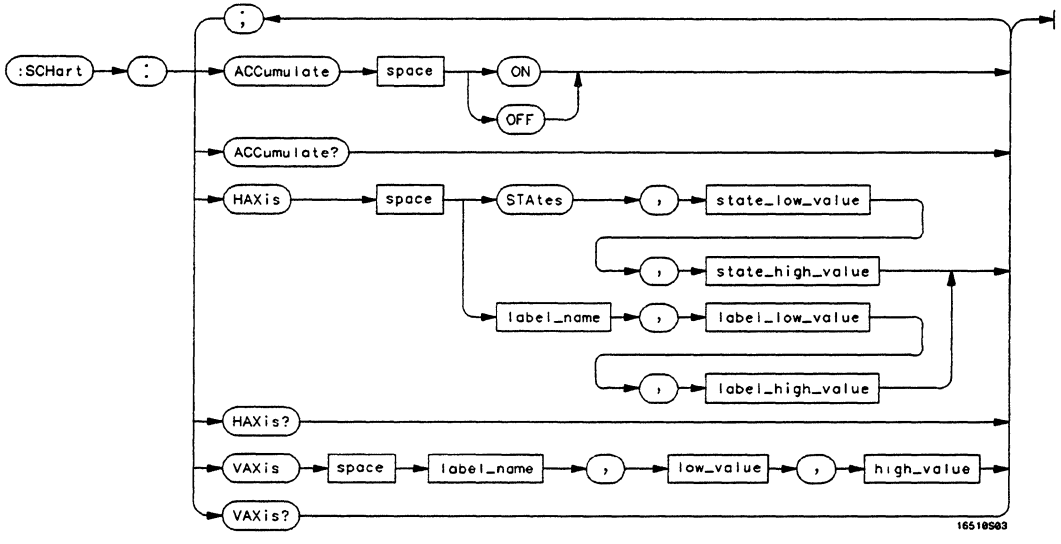
Introduction

The State Chart subsystem provides the commands necessary for programming the Chart display HP 1660A-series logic analyzers . The commands allow you to build charts of label activity, using data normally found in the Listing display. The chart's Y-axis is used to show data values for the label of your choice. The X-axis can be used in two different ways. In one, the X-axis represents states (shown as rows in the State Listing display). In the other, the X-axis represents the data values for another label. When states are plotted along the X-axis, X and O markers are available. Because the State Chart display is simply an alternative way of looking at the data in the State Listing, the X and O markers can be manipulated through the SLISt subsystem. Because the programming commands do not force the menus to switch, you can position the markers in the SLISt subsystem and see the effects in the State Chart display.

The commands in the SCHart subsystem are:

- ACCumulate
- HAXis
- VAXis

Figure 19-1



SCHart Subsystem Syntax Diagram

Table 19-1 **SCHart Parameter Values**

Parameter	Values
state_low_value	integer from –8191 to +8191
state_high_value	integer from <state_low_value> to +8191
label_name	string of up to 6 alphanumeric characters
label_low_value	string from 0 to $2^{32} - 1$ (#HFFFF)
label_high_value	string from <label_low_value> to $2^{32} - 1$ (#HFFFF)
low_value	string from 0 to $2^{32} - 1$ (#HFFFF)
high_value	string from low_value to $2^{32} - 1$ (#HFFFF)

SCHart

Selector

```
:MACHine{1|2}:SCHart
```

The SCHart selector is used as part of a compound header to access the settings found in the State Chart menu. It always follows the MACHine selector because it selects a branch below the MACHine level in the command tree.

Example

```
OUTPUT XXX;":MACHINE1:SCHART:VAXIS 'A', '0', '9'"
```

ACCumulate

Command

```
:MACHine{1|2}:SCHart:ACCumulate {{ON|1}|{OFF|0}}
```

The ACCumulate command allows you to control whether the chart display gets erased between each individual run or whether subsequent waveforms are allowed to be displayed over the previous waveforms.

Example

```
OUTPUT XXX;":MACHINE1:SCHART:ACCUMULATE OFF"
```

Query

```
:MACHine{1|2}:SCHart:ACCumulate?
```

The ACCumulate query returns the current setting. The query always shows the setting as the character "0" (off) or "1" (on).

Returned Format

```
[:MACHine{1|2}:SCHart:ACCumulate] {0|1}<NL>
```

Example

```
OUTPUT XXX;":MACHINE1:SCHART:ACCUMULATE?"
```

HAXis

Command

```
:MACHine{1|2}:SCHart:HAXis {STAtes,
<state_low_value>,<state_high_value>|<label_name>,
<label_low_value>,<label_high_value>}
```

The HAXis command allows you to select whether states or a label's values will be plotted on the horizontal axis of the chart. The axis is scaled by specifying the high and low values.

> The shortform for STATES is STA. This is an intentional deviation from the normal truncation rule.

`<state_low_value>`	integer from −8191 to +8191
`<state_high_value>`	integer from `<state_low_value>` to +8191
`<label_name>`	string of up to 6 alphanumeric characters
`<label_low_value>`	string from 0 to $2^{32}-1$ (#HFFFF)
`<label_high_value>`	string from `<label_low_value>` to $2^{32}-1$ (#HFFFF)

Examples

```
OUTPUT XXX;":MACHINE1:SCHART:HAXIS STATES, -100, 100"
OUTPUT XXX;":MACHINE1:SCHART:HAXIS 'READ', '-511', '511'"
```

Query

```
:MACHine{1|2}:SCHart:HAXis?
```

The HAXis query returns the current horizontal axis label and scaling.

Returned Format

```
[:MACHine{1|2}:SCHart:HAXis] {STAtes,<state_low_value>,
<state_high_value>|<label_name>,<label_low_value>,
<label_high_value>}
```

Example

```
OUTPUT XXX;":MACHINE1:SCHART:HAXIS?"
```

VAXis

Command

```
:MACHine{1|2}:SCHart:VAXis
<label_name>,<low_value>,<high_value>
```

The VAXis command allows you to choose which label will be plotted on the vertical axis of the chart and scale the vertical axis by specifying the high value and low value.

`<label_name>` string of up to 6 alphanumeric characters

`<low_value>` string from 0 to $2^{32}-1$ (#HFFFF)

`<high_value>` string from `<low_value>` to $2^{32}-1$ (#HFFFF)

Examples

```
OUTPUT XXX;":MACHINE2:SCHART:VAXIS 'SUM1', '0', '99'"
OUTPUT XXX;":MACHINE1:SCHART:VAXIS 'BUS', '#H00FF', '#H0500'"
```

Query

```
:MACHine{1|2}:SCHart:VAXis?
```

The VAXis query returns the current vertical axis label and scaling.

Returned Format

```
[:MACHine{1|2}:SCHart:VAXis] <label_name>,<low_value>,
<high_value><NL>
```

Example

```
OUTPUT XXX;":MACHINE1:SCHART:VAXIS?"
```

20

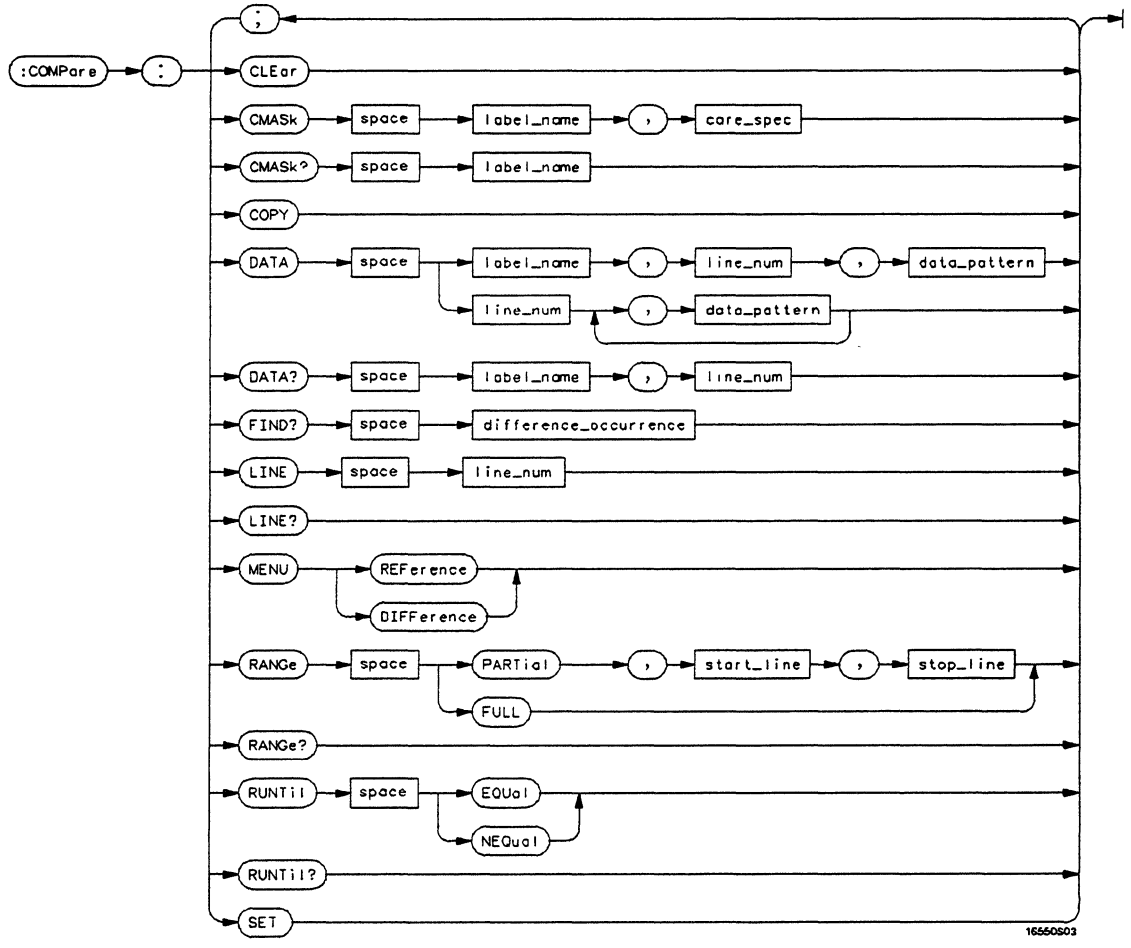
COMPare Subsystem

Introduction

Commands in the state COMPare subsystem provide the ability to do a bit-by-bit comparison between the acquired state data listing and a compare data image. The commands are:

- CLEar
- CMASk
- COPY
- DATA
- FIND
- LINE
- MENU
- RANGe
- RUNTil
- SET

Figure 20-1



COMPare Subsystem Syntax Diagram

Table 20-1 **Compare Parameter Values**

Parameter	Values
label_name	string of up to 6 characters
care_spec	string of characters "{*|.}..."
*	care
.	don't care
line_num	integer from –8191 to +8191
data_pattern	"{B{0|1|X}...| #Q{0|1|2|3|4|5|6|7|X}...| #H{0|1|2|3|4|5|6|7|8|9|A|B|C|D|E|F|X}...| {0|1|2|3|4|5|6|7|8|9}...}"
difference_occurence	integer from 1 to 8192
start_line	integer from –8191 to +8191
stop_line	integer from <start_line> to +8191

COMPare

Selector

```
:MACHine{1|2}:COMPare
```

The COMPare selector is used as part of a compound header to access the settings found in the Compare menu. It always follows the MACHine selector because it selects a branch directly below the MACHine level in the command tree.

Example

```
OUTPUT XXX;":MACHINE1:COMPARE:FIND? 819"
```

CLEar

Command

```
:MACHine{1|2}:COMPare:CLEar
```

The CLEar command clears all "don't cares" in the reference listing and replaces them with zeros except when the CLEar command immediately follows the SET command (see SET command).

Example

```
OUTPUT XXX;":MACHINE2:COMPARE:CLEAR"
```

CMASk

Command

```
:MACHine{1|2}:COMPare:CMASk
<label_name>,<care_spec>
```

The CMASk (Compare Mask) command allows you to set the bits in the channel mask for a given label in the compare listing image to "compares" or "don't compares."

`<label_name>` A string of up to 6 alphanumeric characters

`<care_spec>` A string of characters "{*|.}..." (32 characters maximum)

`<*>` An indicator that tells the logic analyzer that it cares about this bit.

`<.>` An indicator that tells the logic analyzer that it does not care about this bit (don't care).

Example

```
OUTPUT XXX;":MACHINE2:COMPARE:CMASK 'DATA', '*.**..**'"
```

Query

```
:MACHine{1|2}:COMPare:CMASk <label_name>?
```

The CMASk query returns the state of the bits in the channel mask for a given label in the compare listing image.

Returned Format

```
[:MACHine{1|2}:COMPare:CMASk] <label_name>,<care_spec>
```

`<label name>` A string of up to 6 alphanumeric characters

`<care_spec>` A string of characters "{*|.}..." (32 characters maximum)

`<*>` An indicator that tells the logic analyzer that it cares about this bit.

`<.>` An indicator that tells the logic analyzer that it does not care about this bit (don't care).

Example

```
OUTPUT XXX;":MACHINE2:COMPARE:CMASK 'DATA'?"
```

COPY

Command

```
:MACHine{1|2}:COMPare:COPY
```

The COPY command copies the current acquired State Listing for the specified machine into the Compare Reference template. It does not affect the compare range or channel mask settings.

Example

```
OUTPUT XXX;":MACHINE2:COMPARE:COPY"
```

DATA

Command

```
:MACHine{1|2}:COMPare:DATA {<label_name>,
<line_num>,<data_pattern>|<line_num>,
<data_pattern>[, <data_pattern>]... }
```

The DATA command allows you to edit the compare listing image for a given label and state row. When DATA is sent to an instrument where no compare image is defined (such as at power-up) all other data in the image is set to don't cares.

Not specifying the `<label_name>` parameter allows you to write data patterns to more than one label for the given line number. The first pattern is placed in the left-most label, with the following patterns being placed in a left-to-right fashion (as seen on the Compare display). Specifying more patterns than there are labels simply results in the extra patterns being ignored.

Because don't cares (Xs) are allowed in the data pattern, it must always be expressed as a string. You may still use different bases; although, don't cares cannot be used in a decimal number.

`<label_name>` A string of up to 6 alphanumeric characters

`<line_num>` An integer from –8191 to +8191

`<data pattern>` A string in one of the following forms:

```
"{B{0|1|X} . . . |
#Q{0|1|2|3|4|5|6|7|X} . . . |
#H{0|1|2|3|4|5|6|7|8|9|A|B|C|D|E|F|X} . . . |
{0|1|2|3|4|5|6|7|8|9} . . . }"
```

Examples

```
OUTPUT XXX;":MACHINE2:COMPARE:DATA 'CLOCK', 42, '#B011X101X'"
OUTPUT XXX;":MACHINE2:COMPARE:DATA 'OUT3', 0, '#HFF40'"
OUTPUT XXX;":MACHINE1:COMPARE:DATA 129, '#BXX00', '#B1101',
'#B10XX'"
OUTPUT XXX;":MACH2:COMPARE:DATA -511, '4', '64', '16', 256',
'8', '16'"
```

Query

```
:MACHine{1|2}:COMPare:DATA? <label_name>,<line_num>
```

The DATA query returns the value of the compare listing image for a given label and state row.

Returned Format

```
[:MACHine{1|2}:COMPare:DATA] <label_name>,<line_num>,
<data_pattern><NL>
```

`<label_name>` A string of up to 6 alphanumeric characters

`<line_num>` An integer from –8191 to +8191

`<data pattern>` A string in one of the following forms:

```
"{B{0|1|X} . . . |
#Q{0|1|2|3|4|5|6|7|X} . . . |
#H{0|1|2|3|4|5|6|7|8|9|A|B|C|D|E|F|X} . . . |
{0|1|2|3|4|5|6|7|8|9} . . . }"
```

Example

```
10 DIM Label$[6], Response$[80]
15 PRINT "This program shows the values for a signal's Compare
listing"
20 INPUT "Enter signal label: ", Label$
25 OUTPUT XXX;":SYSTEM:HEADER OFF"  !Turn headers off (from
responses)
30 OUTPUT XXX;":MACHINE2:COMPARE:RANGE?"
35 ENTER XXX; First, Last     !Read in the range's end-points
40 PRINT "LINE ", "VALUE of "; Label$
45 FOR State = First TO Last   !Print compare value for each
state
50   OUTPUT XXX;":MACH2:COMPARE:DATA? '" & Label$ & "'," &
VAL$(State)
55   ENTER XXX; Response$
60   PRINT State, Response$
65   NEXT State
70 END
```

FIND

Query

```
:MACHine{1|2}:COMPare:FIND? <difference_occurrence>
```

The FIND query is used to get the line number of a specified difference occurence (first, second, third, etc) within the current compare range, as dictated by the RANGe command (see page 20-11). A difference is counted for each line where at least one of the current labels has a discrepancy between its acquired state data listing and its compare data image.

Invoking the FIND query updates both the Listing and Compare displays so that the line number returned is in the center of the screen.

Returned Format

```
[:MACHine{1|2}:COMPare:FIND] <difference_occurrence>,
<line_number><NL>
```

`<difference_occurrence>` integer from 1 to 8192

`<line_number>` integer from –8191 to +8191

Example

```
OUTPUT XXX;":MACHINE2:COMPARE:FIND? 26"
```

LINE

Command

```
:MACHine{1|2}:COMPare:LINE <line_num>
```

The LINE command allows you to center the compare listing data about a specified line number.

`<line_num>` An integer from –8191 to +8191

Example

```
OUTPUT XXX;":MACHINE2:COMPARE:LINE -511"
```

Query

```
:MACHine{1|2}:COMPare:LINE?
```

The LINE query returns the current line number specified.

Returned Format

```
[:MACHine{1|2}:COMPare:LINE] <line_num>}<NL>
```

`<line_num>` An integer from –8191 to +8191

Example

```
OUTPUT XXX;":MACHINE4:COMPARE:LINE?"
```

MENU

Command

```
:MACHine{1|2}:COMPare:MENU {REFerence|DIFFerence}
```

The MENU command allows you to display the reference or the difference listings in the Compare menu.

Example

```
OUTPUT XXX;":MACHINE2:COMPARE:MENU REFERENCE"
```

RANGe

Command

```
:MACHine{1|2}:COMPare:RANGe
{FULL|PARTial,<start_line>,<stop_line>}
```

The RANGe command allows you to define the boundaries for the comparison. The range entered must be a subset of the lines in the acquire memory.

`<start_line>` integer from –8191 to +8191

`<stop_line>` integer from <start_line> to +8191

Examples

```
OUTPUT XXX;":MACHINE2:COMPARE:RANGE PARTIAL, -511, 512"
OUTPUT XXX;":MACHINE2:COMPARE:RANGE FULL"
```

Query

```
:MACHine{1|2}:COMPare:RANGe?
```

The RANGe query returns the current boundaries for the comparison.

Returned Format

```
[:MACHine{1|2}:COMPare:RANGe] {FULL|PARTial,<start_line>,
<stop_line>}<NL>
```

`<start_line>` integer from –8191 to +8191

`<stop_line>` integer from <start_line> to +8191

Example

```
OUTPUT 707;":MACHINE1:COMPARE:RANGE?"
```

RUNTil

Command

```
:MACHine{1|2}:COMPare:RUNTil {OFF| LT,<value>|GT,
<value>|INRange,<value>,<value>|OUTRange,<value>,<va
lue>|EQUal|NEQual}
```

The RUNTil (run until) command allows you to define a stop condition when the trace mode is repetitive. Specifying OFF causes the analyzer to make runs until either the display's STOP field is touched or the STOP command is issued.

There are four conditions based on the time between the X and O markers. Using this difference in the condition is effective only when time tags have been turned on (see the TAG command in the STRace subsystem). These four conditions are as follows:

- The difference is less than (LT) some value.
- The difference is greater than (GT) some value.
- The difference is inside some range (INRange).
- The difference is outside some range (OUTRange).

End points for the INRange and OUTRange should be at least 8 ns apart since this is the minimum time resolution of the time tag counter.

There are two conditions which are based on a comparison of the acquired state data and the compare data image. You can run until one of the following conditions is true:

- Every channel of every label has the same value (EQUal).
- Any channel of any label has a different value (NEQual).

> The RUNTil instruction (for state analysis) is available in both the SLISt and COMPare subsystems.

`<value>` real number from −9E9 to +9E9

Example

```
OUTPUT XXX;":MACHINE2:COMPARE:RUNTIL EQUAL"
```

Query

```
:MACHine{1|2}:COMPare:RUNTil?
```

The RUNTil query returns the current stop criteria for the comparison when running in repetitive trace mode.

Returned Format

```
[:MACHine{1|2}:COMPare:RUNTil] {OFF| LT,<value>|GT,<value>|
INRange,<value>,<value>|OUTRange,<value>,<value>|EQUal|NEQual}
<NL>
```

<value> real number from −9E9 to +9E9

Example

```
OUTPUT XXX;":MACHINE2:COMPARE:RUNTIL?"
```

SET

Command

```
:MACHine{1|2}:COMPare:SET
```

The SET command sets every state in the reference listing to "don't cares." If you send the SET command by mistake you can immediately send the CLEar command to restore the previous data. This is the only time the CLEar command will not replace "don't cares" with zeros.

Example

```
OUTPUT XXX;":MACHINE2:COMPARE:SET"
```

21

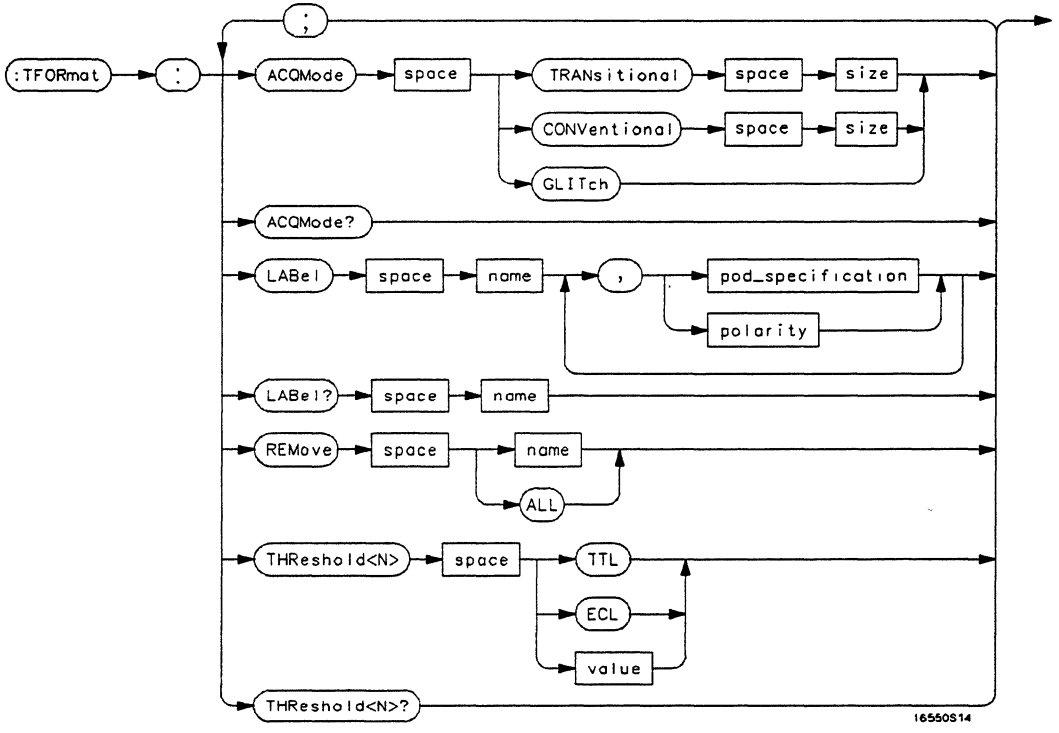
TFORmat Subsystem

Introduction

The TFORmat subsystem contains the commands available for the Timing Format menu in the HP 1660-series logic analyzers. These commands are:

- ACQMode
- LABel
- REMove
- THReshold

Figure 21-1



TFORmat Subsystem Syntax Diagram

Table 21-1 **TFORmat Paramter Values**

Parameter	Values
size	{FULL|HALF}
<N>	{1|2|3|4|5|6|7|8}
name	string of up to 6 alphanumeric characters
polarity	{POSitive|NEGative}
pod_specification	format (integer from 0 to 65535) for a pod (pods are assigned in decreasing order)
value	voltage (real number) −6.00 to +6.00

TFORmat

Selector

```
:MACHine{1|2}:TFORmat
```

The TFORmat selector is used as part of a compound header to access those settings normally found in the Timing Format menu. It always follows the MACHine selector because it selects a branch directly below the MACHine level in the language tree.

Example

```
OUTPUT XXX;":MACHINE1:TFORMAT:ACQMODE?"
```

ACQMode

Command

```
:MACHine{1|2}:TFORmat:ACQMode {TRANSitional
<size>|CONVentional <size>|GLITch}
```

The ACQMode (acquisition mode) command allows you to select the acquisition mode for the timing analyzer. The options are:

- conventional mode at full-channel 250 MHz
- conventional mode at half-channel 500 Mhz
- transitional mode at full-channel 125 MHz
- transitional mode at half-channel 250 MHz
- glitch mode.

`<size>` `{FULL|HALF}`

Example

```
OUTPUT XXX;":MACHINE2:TFORMAT:ACQMODE TRANSITIONAL HALF"
```

Query

```
:MACHine{1|2}:TFORmat:ACQMode?
```

The ACQMode query returns the current acquisition mode.

Returned Format

```
[:MACHine{1|2}:TFORmat:ACQMode] {TRANSitional
<size>|CONVentional <size>|GLITch}<NL>
```

`<size>` `{FULL|HALF}`

Example

```
OUTPUT XXX;":MACHINE2:TFORMAT:ACQMODE?"
```

LABel

Command

```
:MACHine{1|2}:Tformat:LABel <name>,[<polarity>,
<clock_bits>, <upper_bits>, <lower_bits>
[,<upper_bits>,<lower_bits>]...]
```

The LABel command allows you to specify polarity and to assign channels to new or existing labels. If the specified label name does not match an existing label name, a new label will be created.

The order of the pod-specification parameters is significant. The first one listed will match the highest numbered pod assigned to the machine you're using. Each pod specification after that is assigned to the next highest numbered pod. This way they match the left-to-right descending order of the pods you see on the Format display. Not including enough pod specifications results in the lowest numbered pods being assigned a value of zero (all channels excluded). If you include more pod specifications than there are pods for that machine, the extra ones will be ignored. However, an error is reported anytime more than 13 pod specifications are listed.

The polarity can be specified at any point after the label name.

Because pods contain 16 channels, the format value for a pod must be between 0 and 65535 ($2^{16}-1$). When giving the pod assignment in binary (base 2), each bit will correspond to a single channel. A "1" in a bit position means the associated channel in that pod is assigned to that pod and bit. A "0" in a bit position means the associated channel in that pod is excluded from the label. For example, assigning #B1111001100 is equivalent to entering "......****..**.." from the front panel.

A label can not have a total of more than 32 channels assigned to it.

`<name>` string of up to 6 alphanumeric characters

`<polarity>` `{POSitive|NEGative}`

`<clock_bits>` format (integer from 0 to 63) for a clock (clocks are assigned in decreasing order)

`<upper_bits>` format (integer from 0 to 65535) for a pod (pods are assigned in decreasing order)

`<lower_bits>` format (integer from 0 to 65535) for a pod (pods are assigned in decreasing order)

Examples

```
OUTPUT XXX;":MACHINE2:TFORMAT:LABEL 'STAT', POSITIVE,
0,127,40312"
OUTPUT XXX;":MACHINE2:TFORMAT:LABEL 'SIG 1',
#B11,#B0000000011111111,
#B0000000000000000 "
```

Query

```
:MACHine{1|2}:Tformat:LABel? <name>
```

The LABel query returns the current specification for the selected (by name) label. If the label does not exist, nothing is returned. Numbers are always returned in decimal format.

Returned Format

```
[:MACHine{1|2}:Tformat:LABel] <name>,<polarity>[,
<assignment>]...<NL>
```

`<name>` string of up to 6 alphanumeric characters

`<polarity>` `{POSitive|NEGative}`

Example

```
OUTPUT XXX;":MACHINE2:TFORMAT:LABEL? 'DATA'"
```

REMove

Command

```
:MACHine{1|2}:TFORmat:REMove {<name>|ALL}
```

The REMove command allows you to delete all labels or any one label specified by name for a given machine.

`<name>` string of up to 6 alphanumeric characters

Examples

```
OUTPUT XXX;":MACHINE1:TFORMAT:REMOVE 'A'"
OUTPUT XXX;":MACHINE1:TFORMAT:REMOVE ALL"
```

THReshold

Command

```
:MACHine{1|2}:TFORmat:THReshold<N> {TTL|ECL|<value>}
```

The THReshold command allows you to set the voltage threshold for a given pod to ECL, TTL, or a specific voltage from −6.00 V to +6.00 V in 0.05 volt increments.

`<N>` pod number {1|2|3|4|5|6|7|8}

`<value>` voltage (real number) −6.00 to +6.00

`TTL` default value of +1.6 V

`ECL` default value of −1.3 V

Example

```
OUTPUT XXX;":MACHINE1:TFORMAT:THRESHOLD1 4.0"
```

Query

```
:MACHine{1|2}:TFORmat:THReshold<N>?
```

The THReshold query returns the current threshold for a given pod.

Returned Format

```
[:MACHine{1|2}:TFORmat:THReshold<N>] <value><NL>
```

Example

```
OUTPUT XXX;":MACHINE1:TFORMAT:THRESHOLD2?"
```

22

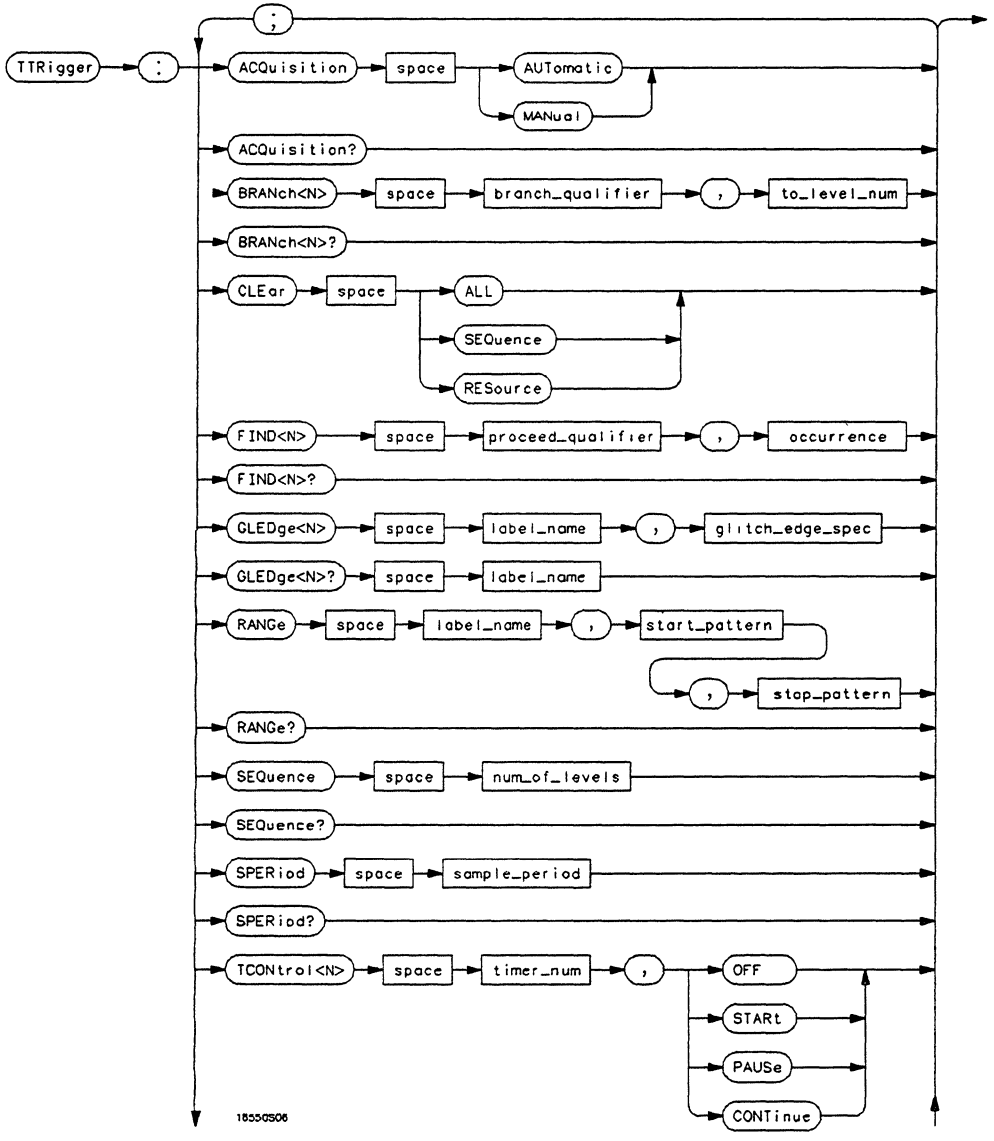
TTRigger (TTRace) Subsystem

Introduction

The TTRigger subsystem contains the commands available for the Timing Trigger menu in the HP 1660-series logic analyzers. The Timing Trigger subsystem will also accept the TTRace selector as used in previous HP 1650-series logic analyzers to eliminate the need to rewrite programs containing TTRace as the selector keyword. The TTRigger subsystem commands are:

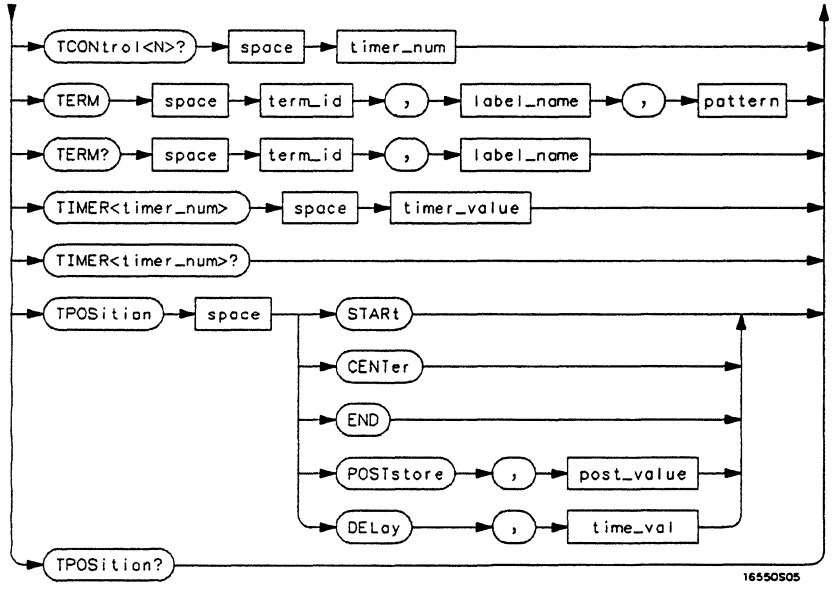
- ACQuisition
- BRANch
- CLEar
- FIND
- GLEDge
- RANGe
- SEQuence
- SPERiod
- TCONtrol
- TERM
- TIMER
- TPOSition

Figure 22-1



TTRigger Subsystem Syntax Diagram

Figure 22-1 (continued)



TTRigger Subsystem Syntax Diagram (continued)

Table 22-1 TTRigger Parameter Values

Parameter	Values
branch_qualifier	<qualifier>
to_lev_num	integer from 1 to last level
proceed_qualifier	<qualifier>
occurrence	number from 1 to 1048575
label_name	string of up to 6 alphanumeric characters
glitch_edge_spec	string consisting of {R|F|E|G|.} R, F, and E represents rising, falling, either edge respectively. G represents a glitch and a period (.) represents a don't care.
start_pattern	"{#B{0|1}...| #Q{0|1|2|3|4|5|6|7}...| #H{0|1|2|3|4|5|6|7|8|9|A|B|C|D|E|F}...| {0|1|2|3|4|5|6|7|8|9}...}"
stop_pattern	"{#B{0|1}...| #Q{0|1|2|3|4|5|6|7}...| #H{0|1|2|3|4|5|6|7|8|9|A|B|C|D|E|F}...| {0|1|2|3|4|5|6|7|8|9}...}"
num_of_levels	integer from 1 to 10
lev_of_trig	integer from 1 to (number of existing sequence levels)
store_qualifier	<qualifier>
state_tag_qualifier	<qualifier>
timer_num	{1|2}
timer_value	400 ns to 500 seconds
term_id	{A|B|C|D|E|F|G|H|I|J}
pattern	"{#B{0|1|X}...| #Q{0|1|2|3|4|5|6|7|X}...| #H{0|1|2|3|4|5|6|7|8|9|A|B|C|D|E|F|X}...| {0|1|2|3|4|5|6|7|8|9}...}"
qualifier	see "Qualifier" on page 22-6
post_store	integer from 0 to 100 representing percentage
time_val	integer from 0 to 500 representing seconds

Qualifier

The qualifier for the timing trigger subsystem can be terms A through J, Timer 1 and 2, and Range 1 and 2. In addition, qualifiers can be the NOT boolean function of terms, timers, and ranges. The qualifier can also be an expression or combination of expressions as shown below and figure 22-2, "Complex Qualifier," on page 22-11.

The following parameters show how qualifiers are specified in all commands of the TTRigger subsystem that use <qualifier>.

<qualifier>	{"ANYSTATE"|"NOSTATE"|"<expression>"}
<expression>	{<expression1a>|<expression1b>|<expression1a> OR <expression1b>|<expression1a> AND <expression1b>}
<expression1a>	{<expression1a_term>|(<expression1a_term>[OR <expression1a_term>]*)|(<expression1a_term>[AND <expression1a_term>]*)}
<expression1a_term>	{<expression2a>|<expression2b>|<expression2c>| <expression2d>}
<expression1b>	{<expression1b_term>|(<expression1b_term>[OR <expression1b_term>]*)|(<expression1b_term>[AND <expression1b_term>]*)}
<expression1b_term>	{<expression2e>|<expression2f>|<expression2g>| <expression2h>}
<expression2a>	{<term3a>|<term3b>|(<term3a> <boolean_op> <term3b>)}
<expression2b>	{<term3c>|<range3a>|(<term3c> <boolean_op> <range3a>)}
<expression2c>	{<term3d>|<gledge3a|(<term3d> <boolean_op> <gledge3a>)}
<expression2d>	{<term3e>|<timer3a>|(<term3e> <boolean_op> <timer3a>)}
<expression2e>	{<term3f>|<term3g>|(<term3f> <boolean_op> <term3g>)}
<expression2f>	{<term3h>|<range3b>|(<term3h> <boolean_op> <range3b>)}
<expression2g>	{<term3i>|<gledge3b>|(<term3i> <boolean_op> <gledge3b>)}
<expression2h>	{<term3j>|<timer3b>|(<term3e> <boolean_op> <timer3b>)}
<boolean_op>	{AND|NAND|OR|NOR|XOR|NXOR}

```
    <term3a>  {A|NOTA}
    <term3b>  {B|NOTB}
    <term3c>  {C|NOTC}
    <term3d>  {D|NOTD}
    <term3e>  {E|NOTE}
    <term3f>  {F|NOTF}
    <term3g>  {G|NOTG}
    <term3h>  {H|NOTH}
    <term3i>  {I|NOTI}
    <term3j>  {J|NOTJ}
  <range3a>  {IN_RANGE1|OUT_RANGE1}
  <range3b>  {IN_RANGE2|OUT_RANGE2}
 <gledge3a>  {GLEDge1|NOT GLEDge1}
 <gledge3b>  {GLEDge2|NOT GLEDge2}
  <timer3a>  {TIMER1<|TIMER1>}
  <timer3b>  {TIMER2<|TIMER2>}
```

* = is optional such that it can be used zero or more times
+ = must be used at least once and can be repeated

Qualifier Rules

The following rules apply to qualifiers:

- Qualifiers are quoted strings and, therefore, need quotes.
- Expressions are evaluated from left to right.
- Parenthesis are used to change the order evaluation and, therefore, are optional.
- An expression must map into the combination logic presented in the combination pop-up menu within the TTRigger menu.

Examples

```
'A'
'( A OR B )'
'(( A OR B ) AND C )'
'(( A OR B ) AND C AND IN_RANGE2 )'
'(( A OR B ) AND ( C AND IN_RANGE1 ))'
'IN_RANGE1 AND ( A OR B ) AND C'
```

TTRigger (TTRace)

Selector

```
:MACHine{1|2}:TTRigger
```

The TTRigger (TTRace) (Trace Trigger) selector is used as a part of a compound header to access the settings found in the Timing Trace menu. It always follows the MACHine selector because it selects a branch directly below the MACHine level in the command tree.

Example

```
OUTPUT XXX;":MACHINE1:TTRIGGER:TAG TIME"
```

ACQuisition

Command

```
:MACHine{1|2}:TTRigger:ACQuisition
{AUTOmatic|MANual}
```

The ACQuisition command allows you to specify the acquisition mode for the Timing analyzer.

Example

```
OUTPUT XXX;":MACHINE1:TTRIGGER:ACQUISITION AUTOMATIC"
```

Query

```
:MACHine{1|2}:TTRigger:ACQuisition?
```

The ACQuisition query returns the current acquisition mode specified.

Returned Format

```
[:MACHine{1|2}:TTRigger:ACQuisition] {AUTOmatic|MANual}<NL>
```

Example

```
OUTPUT XXX;":MACHINE1:TTRIGGER:ACQUISITION?"
```

BRANch

Command

```
:MACHine{1|2}:TTRigger:BRANch<N>
<branch_qualifier>,<to_level_number>
```

The BRANch command defines the branch qualifier for a given sequence level. When this branch qualifier is matched, it will cause the sequencer to jump to the specified sequence level.

The terms used by the branch qualifier (A through J) are defined by the TERM command. The meaning of **IN_RANGE** and **OUT_RANGE** is determined by the RANGE command.

Within the limitations shown by the syntax definitions, complex expressions may be formed using the **AND** and **OR** operators. Expressions are limited to what you could manually enter through the Timing Trigger menu. Regarding parentheses, the syntax definitions on the next page show only the required ones. Additional parentheses are allowed as long as the meaning of the expression is not changed. Figure 22-2, on page 22-11 shows a complex expression as seen in the Timing Trigger menu.

Example

The following statements are all correct and have the same meaning. Notice that the conventional rules for precedence are not followed. The expressions are evaluated from left to right.

```
OUTPUT XXX;":MACHINE1:TTRIGGER:BRANCH1 'C AND D OR F OR G', 1"
OUTPUT XXX;":MACHINE1:TTRIGGER:BRANCH1 '((C AND D)  OR (F OR
G))', 1"
OUTPUT XXX;":MACHINE1:TTRIGGER:BRANCH1 'F OR (C AND D) OR G',1"
```

`<N>`	integer from 1 to `<number_of_levels>`
`<to_level_number>`	integer from 1 to `<number_of_levels>`
`<number_of_levels>`	integer from 1 to the number of existing sequence levels (maximum 10)
`<branch_qualifier>`	`<qualifier>` see "Qualifier" on page 22-6

Examples

```
OUTPUT XXX;":MACHINE1:TTRIGGER:BRANCH1 'ANYSTATE', 3"
OUTPUT XXX;":MACHINE2:TTRIGGER:BRANCH2 'A', 7"
OUTPUT XXX;":MACHINE1:TTRIGGER:BRANCH3 '((A OR B) OR NOTG)', 1"
```

Query Syntax

```
:MACHine{1|2}:TTRigger:BRANch<N>?
```

The BRANch query returns the current branch qualifier specification for a given sequence level.

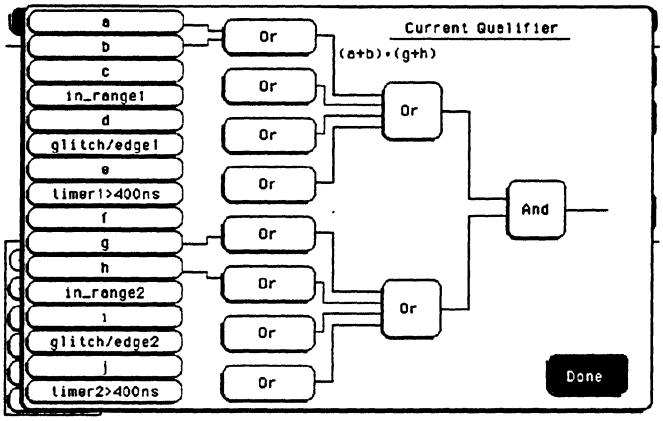
Returned Format

```
[:MACHine{1|2}:TTRigger:BRANch<N>] <branch_qualifier>,
<to_level_num><NL>
```

Example

```
OUTPUT XXX;":MACHINE1:TTRIGGER:BRANCH3?"
```

Figure 22-2



Complex Qualifier

Figure 22-2 is a front-panel representation of the complex qualifier (a OR b) And (g OR h).

Example

This example would be used to specify this complex qualifier.

```
OUTPUT XXX;":MACHINE1:TTRIGGER:BRANCH1 '((A OR B) AND
(G OR H))', 2"
```

Terms A through E, RANGE 1, GLITCH/EDGE1, and TIMER 1 must be grouped together and terms F through J, RANGE 2, GLITCH/EDGE2, and TIMER 2 must be grouped together. In the first level, terms from one group may not be mixed with terms from the other. For example, the expression ((A OR IN_RANGE2) AND (C OR H)) is not allowed because the term C cannot be specified in the E through J group.

In the first level, the operators you can use are ANd, NAND, OR, NOR, XOR, NXOR. Either AND or OR may be used at the second level to join the two groups together. It is acceptable for a group to consist of a single term. Thus, an expression like (B AND G) is legal since the two operands are both simple terms from separate groups.

CLEar

Command

```
:MACHine{1|2}:TTRigger:CLEar
{All|SEQuence|RESource}
```

The CLEar command allows you to clear all settings in the Timing Trigger menu and replace them with the default, clear only the sequence levels, or clear only the resource term patterns.

Example

```
OUTPUT XXX;":MACHINE1:TTRIGGER:CLEAR RESOURCE"
```

FIND

Command

```
:MACHine{1|2}:TTRigger:FIND<N>
<time_qualifier>,<condition_mode>
```

The FIND command defines the time qualifier for a given sequence level. The qualifier tells the timing analyzer when to proceed to the next sequence level. When this proceed qualifier is matched the specified number of times, the sequencer will proceed to the next sequence level. In the sequence level where the trigger is specified, the FIND command specifies the trigger qualifier (see SEQuence command).

The terms A through J are defined by the TERM command. The meaning of **IN_RANGE** and **OUT_RANGE** is determined by the **RANGe** command. Expressions are limited to what you could manually enter through the Timing Trigger menu. Regarding parentheses, the syntax definitions below show only the required ones. Additional parentheses are allowed as long as the meaning of the expression is not changed. See figure 22-2 on page 22-11 for a detailed example.

<N>	integer from 1 to the number of existing sequence levels (maximum 10)
<condition_mode>	{{GT|LT}, <duration_time>|OCCurrence, <occurrence>}
GT	greater than
LT	less than
<duration_time>	real number from 8 ns to 5.00 seconds depending on sample period
<occurrence>	integer from 1 to 1048575
<time_qualifier>	<qualifier> see "Qualifier" on page 22-6

Examples

```
OUTPUT XXX;":MACHINE1:TTRIGGER:FIND1 'ANYSTATE', GT, 10E-6"
OUTPUT XXX;":MACHINE1:TTRIGGER:FIND3 '((NOTA AND NOTB) OR G)',
OCCURRENCE, 10"
```

Query

```
:MACHine{1|2}:TTRigger:FIND4?
```

The FIND query returns the current time qualifier specification for a given sequence level.

Returned Format

```
[:MACHine{1|2}:TTRigger:FIND<N>] <condition_mode>,
<occurrence><NL>
```

Example

```
OUTPUT XXX;":MACHINE1:TTRIGGER:FIND<N>?"
```

GLEDge

Command

```
:MACHine{1|2}:TTRigger:GLEDge<N> <label_name>,
<glitch_edge_spec>
```

The GLEDge (glitch/edge) command allows you to define edge and glitch specifications for a given label. Edge specifications can be `R` (rising), `F` (falling), `E` (either), or "`.`" (don't care). Glitch specifications consist of `G` (glitch) or "`.`" (don't care). Edges and glitches are sent in the same string with the right most string character specifying what the right most bit will be.

> The <glitch_edge_spec> string length must match the exact number of bits assigned to the specified label. If the string length does not match the number of bits, the "Parameter string invalid" message is displayed.

`<N>` `{1|2}`

`<label_name>` string of up to 6 alphanumeric characters

`<glitch_edge_spec>` string consisting of `{R|F|E|G|.| [to total number of bits]}`

Example

For 8 bits assigned and no glitch:

```
OUTPUT XXX;":MACHINE1:TTRIGGER:GLEDGE1 'DATA', '....F..E'"
```

For 16 bits assigned with glitch:

```
OUTPUT XXX;":MACHINE1:TTRIGGER:GLEDGE1 'DATA',
'....GGG.....F..R'"
```

Query

```
:MACHine{1|2}:TTRigger:GLEDe<N>? <label_name>
```

The GLEDge query returns the current specification for the given label.

Returned Format

```
[:MACHine{1|2}:TTRigger:GLEDe<N>]
<label_name>,<glitch_edge_pattern><NL>
```

Example

```
OUTPUT XXX;":MACHINE1:TTRIGGER:GLEDGE1? 'DATA'"
```

RANGe

Command

```
:MACHine{1|2}:TTRigger:RANGE <label_name>,
<start_pattern>,<stop_pattern>
```

The RANGe command allows you to specify a range recognizer term for the specified machine. Since a range can only be defined across one label and, since a label must contain 32 or less bits, the value of the start pattern or stop pattern will be between $(2^{32})-1$ and 0.

> Since a label can only be defined across a maximum of two pods, a range term is only available across a single label; therefore, the end points of the range cannot be split between labels.

When these values are expressed in binary, they represent the bit values for the label at one of the range recognizers' end points. Don't cares are not allowed in the end point pattern specifications.

`<label_name>` string of up to 6 alphanumeric characters

`<start_pattern>`
```
"{#B{0|1}...|
#Q{0|1|2|3|4|5|6|7}...|
#H{0|1|2|3|4|5|6|7|8|9|A|B|C|D|E|F}...|
{0|1|2|3|4|5|6|7|8|9}...}"
```

`<stop_pattern>`
```
"{#B{0|1}...|
#Q{0|1|2|3|4|5|6|7}...|
#H{0|1|2|3|4|5|6|7|8|9|A|B|C|D|E|F}...|
{0|1|2|3|4|5|6|7|8|9}...}"
```

Examples

```
OUTPUT XXX;":MACHINE1:TTRIGGER:RANGE 'DATA', '127', '255' "
OUTPUT XXX;":MACHINE1:TTRIGGER:RANGE 'ABC', '#B00001111',
'#HCF' "
```

Query `:MACHine{1|2}:TTRigger:RANGe?`

The RANGe query returns the range recognizer end point specifications for the range.

Returned Format

```
[:MACHine{1|2}:STRAce:RANGe] <label_name>,<start_pattern>,
<stop_pattern><NL>
```

Example

```
OUTPUT XXX;":MACHINE1:TTRIGGER:RANGE?"
```

SEQuence

Command

```
:MACHine{1|2}:TTRigger:SEQuence <number_of_levels>
```

The SEQuence command defines the timing analyzer trace sequence. First, it deletes the current trace sequence. Then, it inserts the number of levels specified, with default settings. The number of levels can be between 1 and 10 when the analyzer is armed by the RUN key.

`<number_of_levels>` integer from 1 to 10

Example

```
OUTPUT XXX;":MACHINE1:TTRIGGER:SEQUENCE 4"
```

Query

```
:MACHine{1|2}:TTRigger:SEQuence?
```

The SEQuence query returns the current sequence specification.

Returned Format

```
[:MACHine{1|2}:TTRigger:SEQuence] <number_of_levels>,
<level_of_trigger><NL>
```

Example

```
OUTPUT XXX;":MACHINE1:TTRIGGER:SEQUENCE?"
```

SPERiod

Command

```
:MACHine{1|2}:TTRigger:SPERiod <sample_period>
```

The SPERiod command allows you to set the sample period of the timing analyzer in the Conventional and Glitch modes. The sample period range depends on the mode selected and is as follows:

- 2 ns to 8 ms for Conventional Half Channel 500 MHz
- 4 ns to 8 ms for Conventional Full Channel 250 MHz
- 4 ns for Transitional Half Channel
- 8 ns for Transitional Full Channel
- 8 ns to 8 ms for Glitch Half Channel 125 MHz

`<sample_period>` real number from 2 ns to 8 ms depending on mode

Example

```
OUTPUT XXX;":MACHINE1:TTRIGGER:SPERIOD 50E-9"
```

Query

```
:MACHine{1|2}:TTRigger:SPERiod?
```

The SPERiod query returns the current sample period.

Returned Format

```
[:MACHine{1|2}:TTRigger:SPERiod] <sample_period><NL>
```

`<sample_period>` real number from 2 ns to 8 ms depending on mode

Example

```
OUTPUT XXX;":MACHINE1:TTRIGGER:SPERIOD?"
```

TCONtrol

Command

```
:MACHine{1|2}:TTRigger:TCONtrol<N> <timer_num>,
{OFF|STARt|PAUSe|CONTinue}
```

The TCONtrol (timer control) command allows you to turn off, start, pause, or continue the timer for the specified level. The time value of the timer is defined by the TIMER command.

`<N>` integer from 1 to the number of existing sequence levels (maximum 10)

`<timer_num>` `{1|2}`

Example

```
OUTPUT XXX;":MACHINE2:TTRIGGER:TCONTROL6 1, PAUSE"
```

Query

```
:MACHine{1|2}:TTRigger:TCONTROL<N>? <timer_num>
```

The TCONtrol query returns the current TCONtrol setting of the specified level.

Returned Format

```
[:MACHine{1|2}:TTRigger:TCONTROL<N> <timer_num>]
{OFF|STARt|PAUSe|CONTinue}<NL>
```

Example

```
OUTPUT XXX;":MACHINE2:TTRIGGER:TCONTROL6? 1"
```

TERM

Command

```
:MACHine{1|2}:TTRigger:TERM
<term_id>,<label_name>,<pattern>
```

The TERM command allows you to a specify a pattern recognizer term in the specified machine. Each command deals with only one label in the given term; therefore, a complete specification could require several commands. Since a label can contain 32 or less bits, the range of the pattern value will be between $2^{32} - 1$ and 0. When the value of a pattern is expressed in binary, it represents the bit values for the label inside the pattern recognizer term. Since the pattern parameter may contain don't cares and be represented in several bases, it is handled as a string of characters rather than a number.

All 10 terms (A through J) are available to either machine but not both simultaneously. If you send the TERM command to a machine with a term that has not been assigned to that machine, an error message "Legal command but settings conflict" is returned.

`<term_id>` `{A|B|C|D|E|F|G|H|I|J}`

`<label_name>` string of up to 6 alphanumeric characters

`<pattern>`

```
"{#B{0|1|X} . . . |
#Q{0|1|2|3|4|5|6|7|X} . . . |
#H{0|1|2|3|4|5|6|7|8|9|A|B|C|D|E|F|X} . . . |
{0|1|2|3|4|5|6|7|8|9} . . . }"
```

Example

```
OUTPUT XXX;":MACHINE1:TTRIGGER:TERM A,'DATA','255' "
OUTPUT XXX;":MACHINE1:TTRIGGER:TERM B,'ABC','#BXXXX1101' "
```

Query

```
:MACHine{1|2}:TTRigger:TERM? <term_id>,<label_name>
```

The TERM query returns the specification of the term specified by term identification and label name.

Returned Format

```
[:MACHine{1|2}:STRAce:TERM] <term_id>,<label_name>,
<pattern><NL>
```

Example

```
OUTPUT XXX;":MACHINE1:TTRIGGER:TERM? B,'DATA' "
```

TIMER

Command

```
:MACHine{1|2}:TTRigger:TIMER{1|2} <time_value>
```

The TIMER command sets the time value for the specified timer. The limits of the timer are 400 ns to 500 seconds in 16 ns to 500 μs increments. The increment value varies with the time value of the specified timer.

`<time_value>` real number from 400 ns to 500 seconds in increments which vary from 16 ns to 500 μs.

Example

```
OUTPUT XXX;":MACHINE1:TTRIGGER:TIMER1 100E-6"
```

Query

```
:MACHine{1|2}:TTRigger:TIMER{1|2}?
```

The TIMER query returns the current time value for the specified timer.

Returned Format

```
[:MACHine{1|2}:TTRigger:TIMER{1|2}] <time_value><NL>
```

Example

```
OUTPUT XXX;":MACHINE1:TTRIGGER:TIMER1?"
```

TPOSition

Command

```
:MACHine{1|2}:TTRigger:TPOSition
{STARt|CENTer|END|DELay, <time_val>|
POSTstore,<poststore>}
```

The TPOSition (trigger position) command allows you to set the trigger at the start, center, end or at any position in the trace (poststore). Poststore is defined as 0 to 100 percent with a poststore of 100 percent being the same as start position and a poststore 0 percent being the same as an end trace.

`<time_val>` real number from either (2 × sample period) or 16 ns whichever is greater to (1048575 × sample period).

`<poststore>` integer from 0 to 100 representing percentage of poststore.

Examples

```
OUTPUT XXX;":MACHINE1:TTRIGGER:TPOSITION END"
OUTPUT XXX;":MACHINE1:TTRIGGER:TPOSITION POSTstore,75"
```

Query

```
:MACHine{1|2}:TTRigger:TPOSition?
```

The TPOSition query returns the current trigger position setting.

Returned Format

```
[:MACHine{1|2}:TTRigger:TPOSition] {STARt|CENTer|END|DELay,
<time_val>|POSTstore,<poststore>}<NL>
```

Example

```
OUTPUT XXX;":MACHINE1:TTRIGGER:TPOSITION?"
```

23

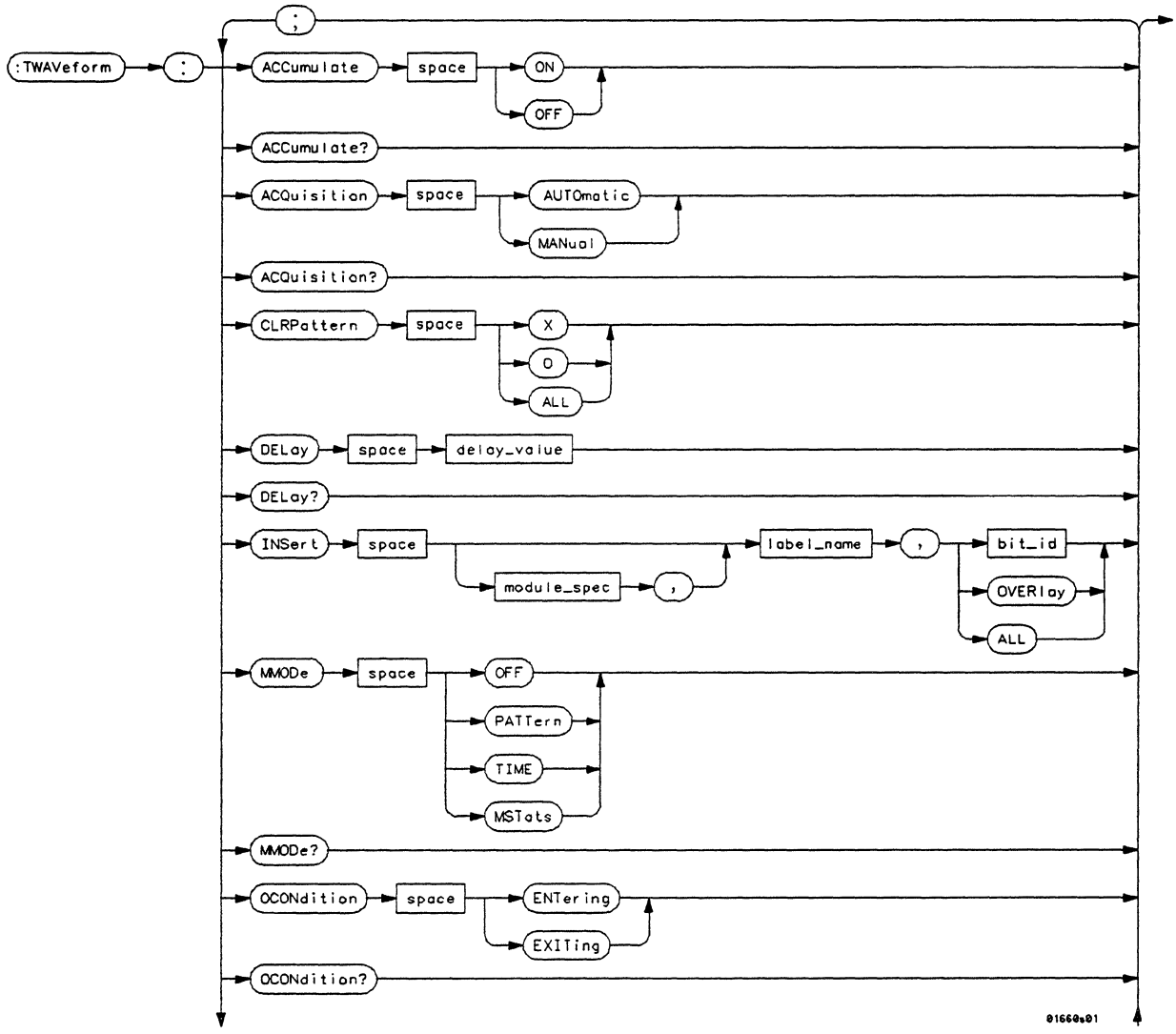
TWAVeform Subsystem

Introduction

The TWAVeform subsystem contains the commands available for the Timing Waveforms menu in the HP 1660A. These commands are:

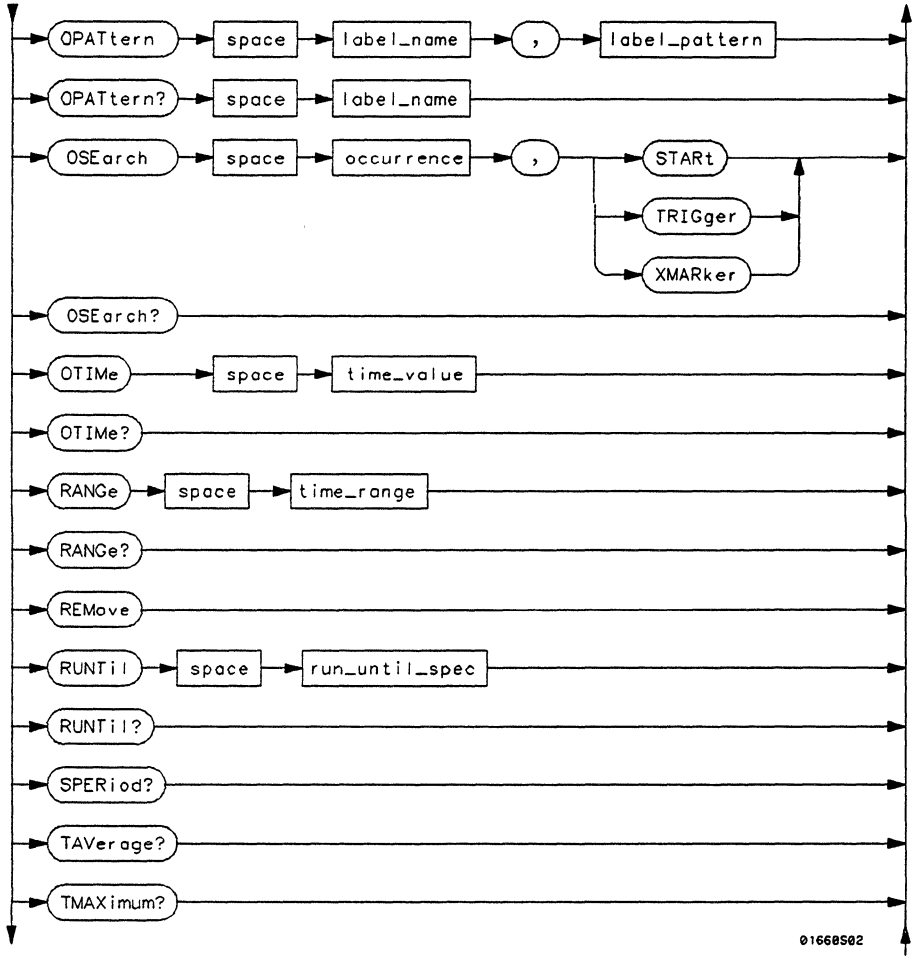
- ACCumulate
- ACQuisition
- CENter
- CLRPattern
- CLRStat
- DELay
- INSert
- MMODe
- OCONdition
- OPATtern
- OSEarch
- OTIMe
- RANGe
- REMove
- RUNTil
- SPERiod
- TAVerage
- TMAXimum
- TMINimum
- TPOSition
- VRUNs
- XCONdition
- XOTime
- XPATtern
- XSEarch
- XTIMe

Figure 23-1



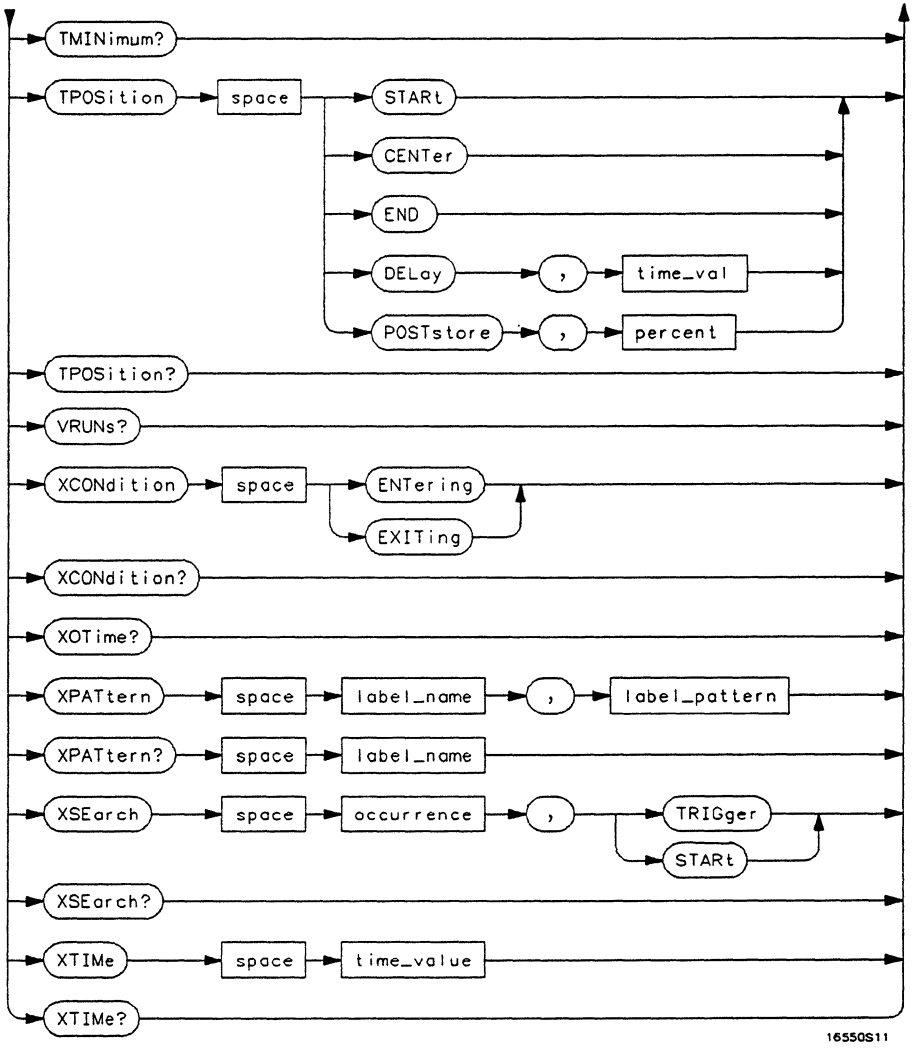
TWAVeform Subsystem Syntax Diagram

Figure 23-1 (continued)



TWAVeform Subsystem Syntax Diagram (continued)

Figure 23-1 (continued)



TWAVeform Subsystem Syntax Diagram (continued)

Table 23-1 **TWAVeform Parameter Values**

Parameter	Value
delay_value	real number between −2500 s and +2500 s
module_spec	{1|2|3|4|5|6|7|8|9|10} 2 through 10 unused
bit_id	integer from 0 to 31
waveform	string containing <acquisition_spec>{1|2}
acquisition_spec	{A|B|C|D|E|F|G|H|I|J} (slot where acquisition card is located)
label_name	string of up to 6 alphanumeric characters
label_pattern	"{#B{0|1|X}...| #Q{0|1|2|3|4|5|6|7|X}...| #H{0|1|2|3|4|5|6|7|8|9|A|C|D|E|F|X}...| {0|1|2|3|4|5|6|7|8|9|X}...}"
occurrence	integer
time_value	real number
label_id	string of one alpha and one numeric character
module_num	slot number in which the time base card is installed
time_range	real number between 10 ns and 10 ks
run_until_spec	{OFF|LT,<value>|GT,<value>|INRange<value>, <value>|OUTRange<value>,<value>}
GT	greater than
LT	less than
value	real number
time_val	real number from 0 to 500 representing seconds

TWAVeform

Selector

```
:MACHine{1|2}:TWAVeform
```

The TWAVeform selector is used as part of a compound header to access the settings found in the Timing Waveforms menu. It always follows the MACHine selector because it selects a branch below the MACHine level in the command tree.

Example

```
OUTPUT XXX;":MACHINE1:TWAVEFORM:DELAY 100E-9"
```

ACCumulate

Command

```
:MACHine{1|2}:TWAVeform:ACCumulate <setting>
```

The ACCumulate command allows you to control whether the chart display gets erased between each individual run or whether subsequent waveforms are allowed to be displayed over the previous ones.

`<setting>` `{0|OFF}` or `{1|ON}`

Example

```
OUTPUT XXX;":MACHINE1:TWAVEFORM:ACCUMULATE ON"
```

Query

```
:MACHine{1|2}:TWAVeform:ACCumulate?
```

The ACCumulate query returns the current setting. The query always shows the setting as the characters, "0" (off) or "1" (on).

Returned Format

```
[:MACHine{1|2}:TWAVeform:ACCumulate] {0|1}<NL>
```

Example

```
OUTPUT XXX;":MACHINE1:TWAVEFORM:ACCUMULATE?"
```

ACQuisition

Command

```
:MACHine{1|2}:TWAVeform:ACQuisition
{AUTOmatic|MANual}
```

The ACQuisition command allows you to specify the acquisition mode for the state analyzer. The acquisition modes are automatic and manual.

Example

```
OUTPUT XXX;":MACHINE2:TWAVEFORM:ACQUISITION AUTOMATIC"
```

Query

```
MACHine{1|2}:TWAVeform:ACQuisition?
```

The ACQuisition query returns the current acquisition mode.

Returned Format

```
[:MACHine{1|2}:TWAVeform:ACQuisition] {AUTOmatic|MANual}<NL>
```

Example

```
OUTPUT XXX;":MACHINE2:TWAVEFORM:ACQUISITION?"
```

CENTer

Command

```
:MACHine{1|2}:Twaveform:CENTer <marker_type>
```

The CENTer command allows you to center the waveform display about the specified markers.

<marker_type>

```
{X|O|XO|TRIGger}
```

Example

```
OUTPUT XXX;":MACHINE1:TWAVEFORM:CENTER X"
```

CLRPattern

Command

```
:MACHine{1|2}:TWAVeform:CLRPattern {X|O|ALL}
```

The CLRPattern command allows you to clear the patterns in the selected Specify Patterns menu.

Example

```
OUTPUT XXX;":MACHINE1:TWAVEFORM:CLRPATTERN ALL"
```

CLRStat

Command

```
:MACHine{1|2}:Twaveform:CLRStat
```

The CLRStat command allows you to clear the waveform statistics without having to stop and restart the acquisition.

Example

```
OUTPUT XXX;":MACHINE1:TWAVEFORM:CLRSTAT"
```

DELay

Command

```
:MACHine{1|2}:TWAVeform:DELay <delay_value>
```

The DELay command specifies the amount of time between the timing trigger and the horizontal center of the the timing waveform display. The allowable values for delay are −2500 s to +2500 s. If the acquisition mode is automatic, then in glitch acquisition mode, as delay becomes large in an absolute sense, the sample rate is adjusted so that data will be acquired in the time window of interest. In transitional acquisition mode, data may not fall in the time window since the sample period is fixed and the amount of time covered in memory is dependent on how frequent the input signal transitions occur.

`<delay_value>` real number between −2500 s and +2500 s

Example

```
OUTPUT XXX;":MACHINE1:TWAVEFORM:DELAY 100E-6"
```

Query

```
:MACHine{1|2}:TWAVeform:DELay?
```

The DELay query returns the current time offset (delay) value from the trigger.

Returned Format

```
[:MACHine{1|2}:TWAVeform:DELay] <delay_value><NL>
```

Example

```
OUTPUT XXX;":MACHINE1:TWAVEFORM:DELAY?"
```

INSert

Command

```
:MACHine{1|2}:TWAVeform:INSert [<module_spec>,]
<label_name>[,{<bit_id>|OVERlay|ALL}]
```

The INSert command allows you to add waveforms to the state waveform display. Waveforms are added from top to bottom on the screen. When 96 waveforms are present, inserting additional waveforms replaces the last waveform. Bit numbers are zero based, so a label with 8 bits is referenced as bits 0 through 7. Specifying OVERlay causes a composite waveform display of all bits or channels for the specified label. If you do not specify the third parameter, ALL is assumed.

`<module_spec>` `{1|2|3|4|5|6|7|8|9|10}` 2 through 10 unused.

`<label_name>` string of up to 6 alphanumeric characters

`<bit_id>` integer from 0 to 31

Example

```
OUTPUT XXX;":MACHINE1:TWAVEFORM:INSERT 1, 'WAVE',10"
```

MMODe

Command

```
:MACHine{1|2}:TWAVeform:MMODe
{OFF|PATTern|TIME|MSTats}
```

The MMODe (Marker Mode) command selects the mode controlling marker movement and the display of the marker readouts. When PATTern is selected, the markers will be placed on patterns. When TIME is selected, the markers move on time. In MSTats, the markers are placed on patterns, but the readouts will be time statistics.

Example

```
OUTPUT XXX; ":MACHINE1:TWAVEFORM:MMODE TIME"
```

Query

```
:MACHine{1|2}:TWAVeform:MMODe?
```

The MMODe query returns the current marker mode.

Returned Format

```
[:MACHine{1|2}:TWAVeform:MMODe] <marker_mode><NL>
```

<marker_mode>

```
{OFF|PATTern|TIME|MSTats}
```

Example

```
OUTPUT XXX;":MACHINE1:TWAVEFORM:MMODE?"
```

OCONdition

Command

```
:MACHine{1|2}:TWAVeform:OCONdition
{ENTering|EXITing}
```

The OCONdition command specifies where the O marker is placed. The O marker can be placed on the entry or exit point of the OPATtern when in the PATTern marker mode.

Example

```
OUTPUT XXX; ":MACHINE1:TWAVEFORM:OCONDITION ENTERING"
```

Query

```
:MACHine{1|2}:TWAVeform:OCONdition?
```

The OCONdition query returns the current setting.

Returned Format

```
[:MACHine{1|2}:TWAVeform:OCONdition] {ENTering|EXITing}<NL>
```

Example

```
OUTPUT XXX;":MACHINE1:TWAVEFORM:OCONDITION?"
```

OPATtern

Command

```
:MACHine{1|2}:TWAVeform:OPATtern
<label_name>,<label_pattern>
```

The OPATtern command allows you to construct a pattern recognizer term for the O marker which is then used with the OSEarch criteria and OCONdition when moving the marker on patterns. Since this command deals with only one label at a time, a complete specification could require several invocations.

When the value of a pattern is expressed in binary, it represents the bit values for the label inside the pattern recognizer term. In whatever base is used, the value must be between 0 and $2^{32} - 1$, since a label may not have more than 32 bits. Because the `<label_pattern>` parameter may contain don't cares, it is handled as a string of characters rather than a number.

`<label_name>` string of up to 6 alphanumeric characters

`<label_pattern>`

```
"{#B{0|1|X} . . . |
#Q{0|1|2|3|4|5|6|7|X} . . . |
#H{0|1|2|3|4|5|6|7|8|9|A|B|C|D|E|F|X} . . . |
{0|1|2|3|4|5|6|7|8|9} . . . }"
```

Example

```
OUTPUT XXX; ":MACHINE1:TWAVEFORM:OPATTERN 'A','511'"
```

Query

```
:MACHine{1|2}:TWAVeform:OPATtern? <label_name>
```

The OPATtern query, in pattern marker mode, returns the pattern specification for a given label name. In the time marker mode, the query returns the pattern under the O marker for a given label. If the O marker is not placed on valid data, don't cares (X) are returned.

Returned Format

```
[:MACHine{1|2}:TWAVeform:OPATtern] <label_name>,
<label_pattern><NL>
```

Example

```
OUTPUT XXX;":MACHINE1:TWAVEFORM:OPATTERN? 'A'"
```

OSEarch

Command

```
:MACHine{1|2}:TWAVeform:OSEarch
<occurrence>,<origin>
```

The OSEarch command defines the search criteria for the O marker which is then used with the associated OPATtern recognizer specification and the OCONdition when moving markers on patterns. The origin parameter tells the marker to begin a search with the trigger or with the X marker. The actual occurrence the marker searches for is determined by the occurrence parameter of the OPATtern recognizer specification, relative to the origin. An occurrence of 0 places a marker on the selected origin. With a negative occurrence, the marker searches before the origin. With a positive occurrence, the marker searches after the origin.

`<origin>` `{STARt|TRIGger|XMARker}`

`<occurrence>` integer from −8192 to +8192

Example

```
OUTPUT XXX; ":MACHINE1:TWAVEFORM:OSEARCH +10,TRIGGER"
```

Query

```
:MACHine{1|2}:TWAVeform:OSEarch?
```

The OSEarch query returns the search criteria for the O marker.

Returned Format

```
[:MACHine{1|2}:TWAVeform:OSEarch] <occurrence>,<origin><NL>
```

Example

```
OUTPUT XXX;":MACHINE1:TWAVEFORM:OSEARCH?"
```

OTIMe

Command `:MACHine{1|2}:TWAVeform:OTIMe <time_value>`

The OTIMe command positions the O marker in time when the marker mode is TIME. If data is not valid, the command performs no action.

`<time_value>` real number −2.5 ks to +2.5 ks

Example

```
OUTPUT XXX; ":MACHINE1:TWAVEFORM:OTIME 30.0E-6"
```

Query `:MACHine{1|2}:TWAVeform:OTIMe?`

The OTIMe query returns the O marker position in time. If data is not valid, the query returns 9.9E37.

Returned Format `[:MACHine{1|2}:TWAVeform:OTIMe] <time_value><NL>`

Example

```
OUTPUT XXX;":MACHINE1:TWAVEFORM:OTIME?"
```

RANGe

Command

```
:MACHine{1|2}:TWAVeform:RANGe <time_value>
```

The RANGe command specifies the full-screen time in the timing waveform menu. It is equivalent to ten times the seconds-per-division setting on the display. The allowable values for RANGe are from 10 ns to 10 ks.

`<time_range>` real number between 10 ns and 10 ks

Example

```
OUTPUT XXX;":MACHINE1:TWAVEFORM:RANGE 100E-9"
```

Query

```
:MACHine{1|2}:TWAVeform:RANGe?
```

The RANGe query returns the current full-screen time.

Returned Format

```
[:MACHine{1|2}:TWAVeform:RANGe] <time_value><NL>
```

Example

```
OUTPUT XXX;":MACHINE1:TWAVEFORM:RANGE?"
```

REMove

Command

```
:MACHine{1|2}:TWAVeform:REMove
```

The REMove command deletes all waveforms from the display.

Example

```
OUTPUT XXX;":MACHINE1:TWAVEFORM:REMOVE"
```

RUNTil

Command

```
:MACHine{1|2}:TWAVeform:RUNTil <run_until_spec>
```

The RUNTil (run until) command defines stop criteria based on the time between the X and O markers when the trace mode is in repetitive. When OFF is selected, the analyzer will run until either the STOP touch screen field is touched, or, the STOP command is sent. Run until time between X and O marker options are:

- Less Than (LT) a specified time value.
- Greater Than (GT) a specified time value.
- In the range (INRange) between two time values.
- Out of the range (OUTRange) between two time values

End points for the INRange and OUTRange should be at least 2 ns apart since this is the minimum time at which data is sampled.

This command affects the timing analyzer only, and has no relation to the RUNTil commands in the SLISt and COMPare subsystems.

`<run_until_spec>` `{OFF|LT,<value>|GT,<value>|INRange<value>,<value>|OUTRange<value>,<value>}`

`<value>` real number

Examples

```
OUTPUT XXX;":MACHINE1:TWAVEFORM:RUNTIL GT, 800.0E-6"
OUTPUT XXX;":MACHINE1:TWAVEFORM:RUNTIL INRANGE, 4.5, 5.5"
```

Query

```
:MACHine{1|2}:TWAVeform:RUNTil?
```

The RUNTil query returns the current stop criteria.

Returned Format

```
[:MACHine{1|2}:TWAVeform:RUNTil] <run_until_spec><NL>
```

Example

```
OUTPUT XXX;":MACHINE1:TWAVEFORM:RUNTIL?"
```

SPERiod

Command

```
:MACHine{1|2}:TWAVeform:SPERiod <sample_period>
```

The SPERiod command allows you to set the sample period of the timing analyzer in the Conventional and Glitch modes. The sample period range depends on the mode selected and is as follows:

- 2 ns to 8 ms for Conventional Half Channel 500 MHz
- 4 ns to 8 ms for Conventional Full Channel 250 MHz
- 8 ns to 8 ms for Glitch Half Channel 125 MHz

`<sample_period>` real number from 2 ns to 8 ms depending on mode

Example

```
OUTPUT XXX;":MACHINE1:TWAVEFORM:SPERIOD 50E-9"
```

Query

```
:MACHine{1|2}:TWAVeform:SPERiod?
```

The SPERiod query returns the current sample period.

Returned Format

```
[:MACHine{1|2}:TWAVeform:SPERiod] <sample_period><NL>
```

Example

```
OUTPUT XXX;":MACHINE1:TWAVEFORM:SPERIOD?"
```

TAVerage

Query

```
:MACHine{1|2}:TWAVeform:TAVerage?
```

The TAVerage query returns the value of the average time between the X and O markers. If there is no valid data, the query returns 9.9E37.

Returned Format

```
[:MACHine{1|2}:TWAVeform:TAVerage] <time_value><NL>
```

`<time_value>` real number

Example

```
OUTPUT XXX;":MACHINE1:TWAVEFORM:TAVERAGE?"
```

TMAXimum

Query

```
:MACHine{1|2}:TWAVeform:TMAXimum?
```

The TMAXimum query returns the value of the maximum time between the X and O markers. If there is no valid data, the query returns 9.9E37.

Returned Format

```
[:MACHine{1|2}:TWAVeform:TMAXimum] <time_value><NL>
```

`<time_value>` real number

Example

```
OUTPUT XXX;":MACHINE1:TWAVEFORM:TMAXIMUM?"
```

TMINimum

Query

```
:MACHine{1|2}:TWAVeform:TMINimum?
```

The TMINimum query returns the value of the minimum time between the X and O markers. If there is no valid data, the query returns 9.9E37.

Returned Format

```
[:MACHine{1|2}:TWAVeform:TMINimum] <time_value><NL>
```

`<time_value>` real number

Example

```
OUTPUT XXX;":MACHINE1:TWAVEFORM:TMINIMUM?"
```

TPOSition

Command

```
:MACHine{1|2}:TWAVeform:TPOSition
{STARt|CENTer|END|DELay,<time_val>|
POSTstore,<percent>}
```

The TPOSition command allows you to control where the trigger point is placed. The trigger point can be placed at the start, center, end, at a percentage of post store, or at a value specified by delay. The post store option is the same as the `User Defined` option when setting the trigger point from the front panel.

The TPOSition command is only available when the acquisition mode is set to manual.

`<time_val>` real number from 0 to 500 seconds

`<percent>` integer from 1 to 100

Example

```
OUTPUT XXX;":MACHINE2:TWAVEFORM:TPOSITION CENTER"
```

Query `:MACHine{1|2}:TWAVeform:TPOSition?`

The TPOSition query returns the current trigger setting.

Returned Format `[:MACHine{1|2}:TWAVeform:TPOSition] {STARt|CENTer|END|DELay, <time_val>|POSTstore,<percent>}<NL>`

`<time_val>` real number from 0 to 500 seconds

Example

```
OUTPUT XXX;":MACHINE2:TWAVEFORM:TPOSition?"
```

VRUNs

Query `:MACHine{1|2}:TWAVeform:VRUNs?`

The VRUNs query returns the number of valid runs and total number of runs made. Valid runs are those where the pattern search for both the X and O markers was successful resulting in valid delta time measurements.

Returned Format `[:MACHine{1|2}:TWAVeform:VRUNs] <valid_runs>,<total_runs><NL>`

`<valid_runs>` zero or positive integer

`<total_runs>` zero or positive integer

Example

```
OUTPUT XXX;":MACHINE1:TWAVEFORM:VRUNS?"
```

XCONdition

Command

```
:MACHine{1|2}:TWAVeform:XCONdition
{ENTering|EXITing}
```

The XCONdition command specifies where the X marker is placed. The X marker can be placed on the entry or exit point of the XPATtern when in the PATTern marker mode.

Example

```
OUTPUT XXX; ":MACHINE1:TWAVEFORM:XCONDITION ENTERING"
```

Query

```
:MACHine{1|2}:TWAVeform:XCONdition?
```

The XCONdition query returns the current setting.

Returned Format

```
[:MACHine{1|2}:TWAVeform:XCONdition] {ENTering|EXITing}<NL>
```

Example

```
OUTPUT XXX;":MACHINE1:TWAVEFORM:XCONDITION?"
```

XOTime

Query

```
:MACHine{1|2}:TWAVeform:XOTime?
```

The XOTime query returns the time from the X marker to the O marker. If data is not valid, the query returns 9.9E37.

Returned Format

```
[:MACHine{1|2}:TWAVeform:XOTime] <time_value><NL>
```

`<time_value>` real number

Example

```
OUTPUT XXX;":MACHINE1:TWAVEFORM:XOTIME?"
```

XPATtern

Command

```
:MACHine{1|2}:TWAVeform:XPATtern <label_name>,
<label_pattern>
```

The XPATtern command allows you to construct a pattern recognizer term for the X marker which is then used with the XSEarch criteria and XCONdition when moving the marker on patterns. Since this command deals with only one label at a time, a complete specification could require several iterations.

When the value of a pattern is expressed in binary, it represents the bit values for the label inside the pattern recognizer term. In whatever base is used, the value must be between 0 and $2^{32} - 1$, since a label may not have more than 32 bits. Because the `<label_pattern>` parameter may contain don't cares, it is handled as a string of characters rather than a number.

`<label_name>` string of up to 6 alphanumeric characters

`<label_pattern>`

```
"{#B{0|1|X} . . . |
#Q{0|1|2|3|4|5|6|7|X} . . . |
#H{0|1|2|3|4|5|6|7|8|9|A|B|C|D|E|F|X} . . . |
{0|1|2|3|4|5|6|7|8|9} . . . }"
```

Example

```
OUTPUT XXX; ":MACHINE1:TWAVEFORM:XPATTERN 'A','511'"
```

Query

```
:MACHine{1|2}:TWAVeform:XPATtern? <label_name>
```

The XPATtern query, in pattern marker mode, returns the pattern specification for a given label name. In the time marker mode, the query returns the pattern under the X marker for a given label. If the X marker is not placed on valid data, don't cares `(X)` are returned.

Returned Format

```
[:MACHine{1|2}:TWAVeform:XPATtern] <label_name>,
<label_pattern><NL>
```

Example

```
OUTPUT XXX;":MACHINE1:TWAVEFORM:XPATTERN? 'A'"
```

XSEarch

Command

```
:MACHine{1|2}:TWAVeform:XSEarch
<occurrence>,<origin>
```

The XSEarch command defines the search criteria for the X marker which is then used with the associated XPATtern recognizer specification and the XCONdition when moving markers on patterns. The origin parameter tells the marker to begin a search with the trigger. The occurrence parameter determines which occurrence of the XPATtern recognizer specification, relative to the origin, the marker actually searches for. An occurrence of 0 (zero) places a marker on the origin.

`<origin>` `{TRIGger|STARt}`

`<occurrence>` integer from −8192 to +8192

Example

```
OUTPUT XXX; ":MACHINE1:TWAVEFORM:XSEARCH,+10,TRIGGER"
```

Query

```
:MACHine{1|2}:TWAVeform:XSEarch?
<occurrence>,<origin>
```

The XSEarch query returns the search criteria for the X marker.

Returned Format

```
[:MACHine{1|2}:TWAVeform:XSEarch] <occurrence>,<origin><NL>
```

Example

```
OUTPUT XXX;":MACHINE1:TWAVEFORM:XSEARCH?"
```

XTIMe

Command

```
:MACHine{1|2}:TWAVeform:XTIMe <time_value>
```

The XTIMe command positions the X marker in time when the marker mode is TIME. If data is not valid, the command performs no action.

`<time_value>` real number from −2.5 ks to +2.5 ks

Example

```
OUTPUT XXX; ":MACHINE1:TWAVEFORM:XTIME 40.0E-6"
```

Query

```
:MACHine{1|2}:TWAVeform:XTIMe?
```

The XTIMe query returns the X marker position in time. If data is not valid, the query returns 9.9E37.

Returned Format

```
[:MACHine{1|2}:TWAVeform:XTIMe] <time_value><NL>
```

Example

```
OUTPUT XXX;":MACHINE1:TWAVEFORM:XTIME?"
```

24

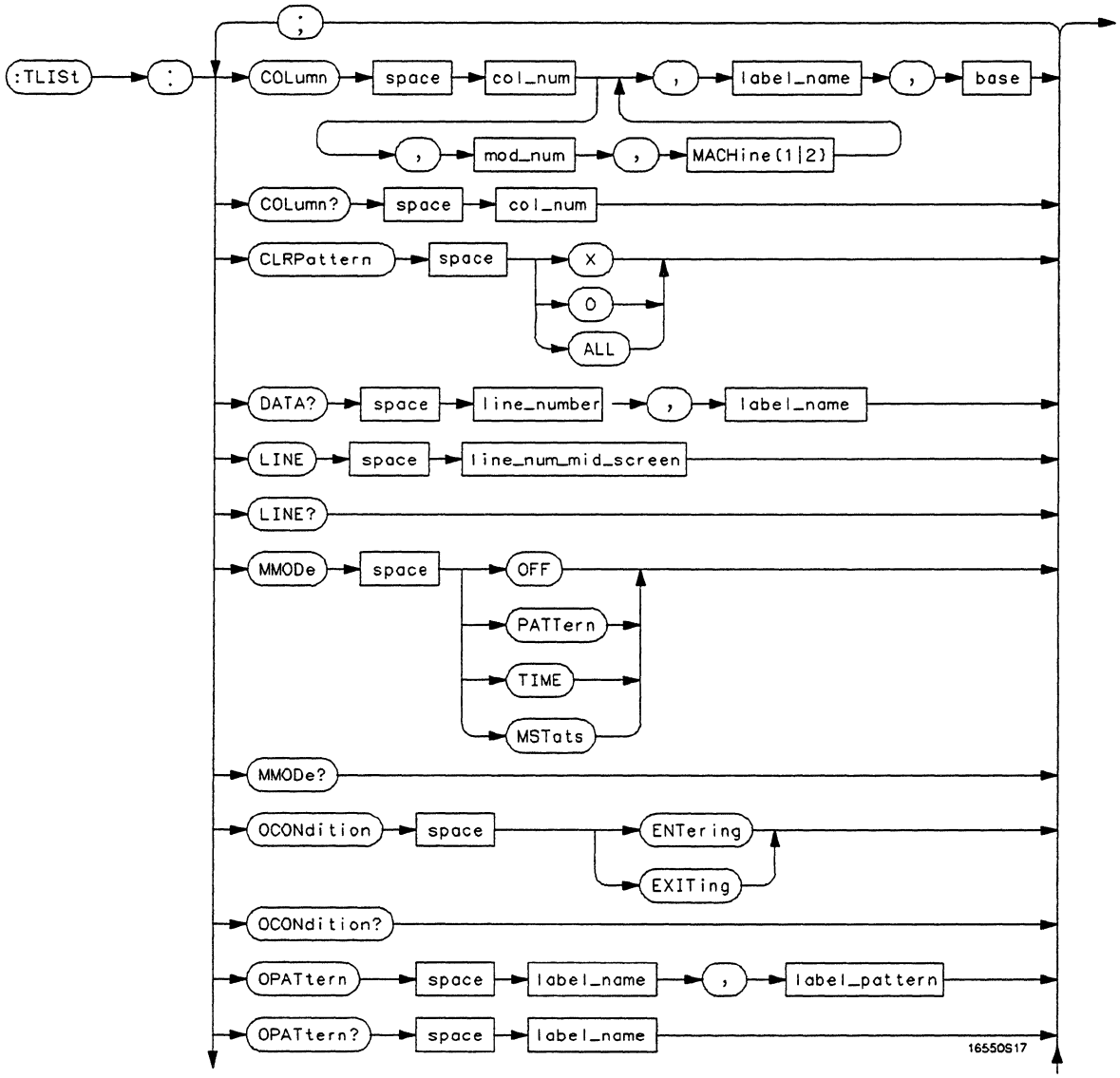
TLISt Subsystem

Introduction

The TLISt subsystem contains the commands available for the Timing Listing menu in the HP 1660-series logic analyzers and is the same as the SLISt subsystem with the exception of the OCONdition and XCONdition commands. The TLISt subsystem commands are:

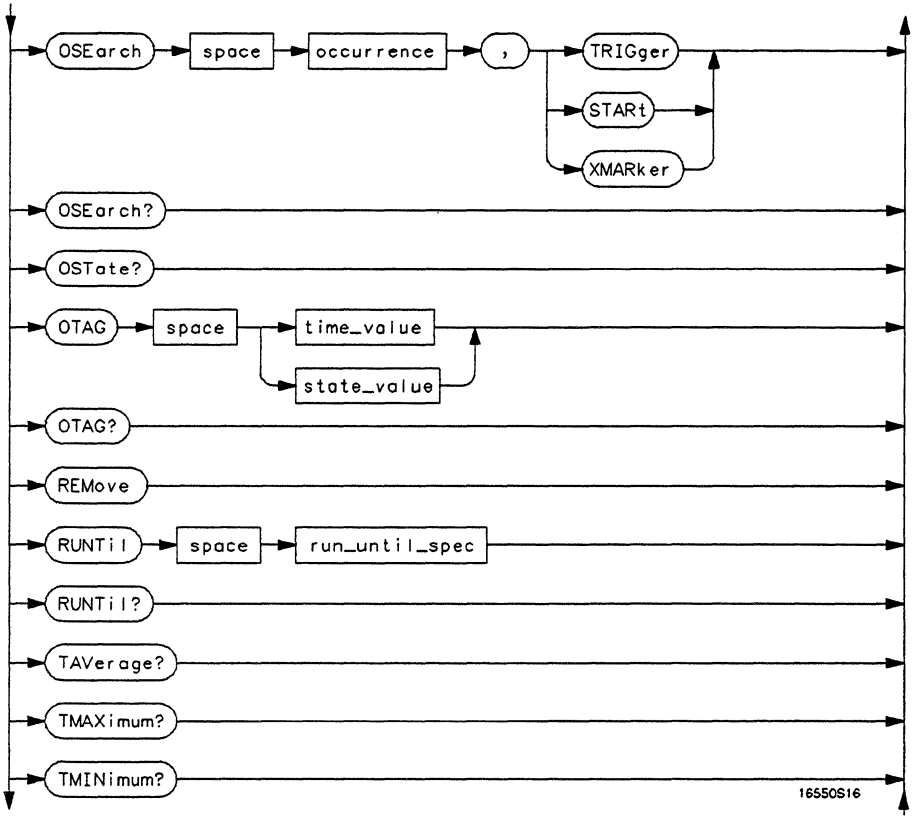
- COLumn
- CLRPattern
- DATA
- LINE
- MMODe
- OCONdition
- OPATtern
- OSEarch
- OSTate
- OTAG
- REMove
- RUNTil
- TAVerage
- TMAXimum
- TMINimum
- VRUNs
- XCONdition
- XOTag
- XOTime
- XPATtern
- XSEarch
- XSTate
- XTAG

Figure 24-1



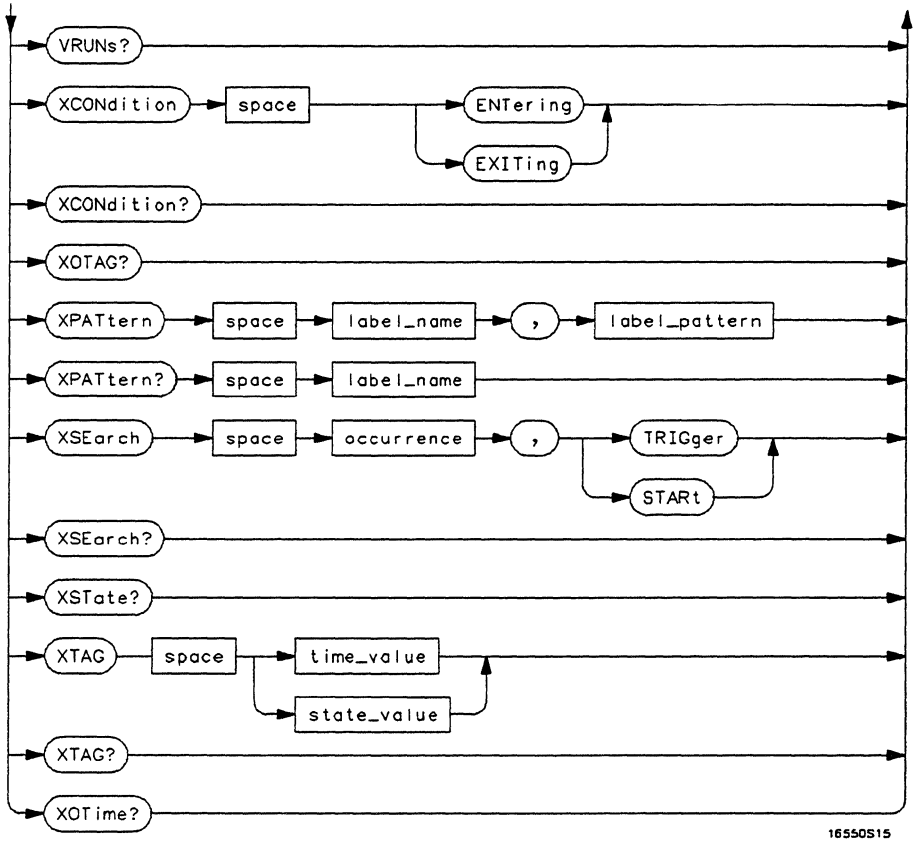
TLISt Subsystem Syntax Diagram

Figure 24-1 (continued)



TLISt Subsystem Syntax Diagram (continued)

Figure 24-1 (continued)



TLISt Subsystem Syntax Diagram (continued)

Table 24-1 TLISt Parameter Values

Parameter	Values
module_num	{1|2|3|4|5|6|7|8|9|10} 2 through 10 not used
mach_num	{1|2}
col_num	integer from 1 to 61
line_number	integer from −8191 to +8191
label_name	string of up to 6 alphanumeric characters
base	{BINary|HEXadecimal|OCTal|DECimal|TWOS| ASCii|SYMBol|IASSembler} for labels or {ABSolute|RELative} for tags
line_num_mid_screen	integer from −8191to +8191
label_pattern	"{#B{0|1|X}...| #Q{0|1|2|3|4|5|6|7|X}...| #H{0|1|2|3|4|5|6|7|8|9|A|B|C|D|E|F|X}...| {0|1|2|3|4|5|6|7|8|9}...}"
occurrence	integer from −8191 to +8191
time_value	real number
state_value	real number
run_until_spec	{OFF|LT,<value>|GT,<value>|INRange,<value>, <value>|OUTRange,<value>,<value>}
value	real number

TLISt

Selector

```
:MACHine{1|2}:TLISt
```

The TLISt selector is used as part of a compound header to access those settings normally found in the Timing Listing menu. It always follows the MACHine selector because it selects a branch directly below the MACHine level in the command tree.

Example

```
OUTPUT XXX;":MACHINE1:TLIST:LINE 256"
```

COLumn

Command

```
:MACHine{1|2}:TLISt:COLumn <col_num>[,<module_num>,
MACHine{1|2}],<label_name>,<base>
```

The COLumn command allows you to configure the timing analyzer list display by assigning a label name and base to one of the 61 vertical columns in the menu. A column number of 1 refers to the left most column. When a label is assigned to a column it replaces the original label in that column.

When the label name is "TAGS," the TAGS column is assumed and the next parameter must specify RELative or ABSolute.

> A label for tags must be assigned in order to use ABSolute or RELative state tagging.

<col_num>	integer from 1 to 61
<module_num>	{1|2|3|4|5|6|7|8|9|10} 2 through 10 unused
<label_name>	a string of up to 6 alphanumeric characters
<base>	{BINary|HEXadecimal|OCTal|DECimal|TWOS|ASCii|SYMBol|IASSembler} for labels or {ABSolute|RELative} for tags

Example

```
OUTPUT XXX;":MACHINE1:TLIST:COLUMN 4,1,'A',HEX"
```

Query

```
:MACHine{1|2}:TLISt:COLumn? <col_num>
```

The COLumn query returns the column number, label name, and base for the specified column.

Returned Format

```
[:MACHine{1|2}:TLISt:COLumn] <col_num>,<module_num>
,MACHine{1|2},<label_name>,<base><NL>
```

Example

```
OUTPUT XXX;":MACHINE1:TLIST:COLUMN? 4"
```

CLRPattern

Command

```
:MACHine{1|2}:TLISt:CLRPattern {X|O|ALL}
```

The CLRPattern command allows you to clear the patterns in the selected Specify Patterns menu.

Example

```
OUTPUT XXX;":MACHINE1:TLIST:CLRPATTERN O"
```

DATA

Query

```
:MACHine{1|2}:TLISt:DATA? <line_number>,
<label_name>
```

The DATA query returns the value at a specified line number for a given label. The format will be the same as the one shown in the Listing display.

Returned Format

```
[:MACHine{1|2}:TLISt:DATA] <line_number>,<label_name>,
<pattern_string><NL>
```

`<line_number>` integer from −8191 to +8191

`<label_name>` string of up to 6 alphanumeric characters

`<pattern_string>`

```
"{#B{0|1|X} . . . |
#Q{0|1|2|3|4|5|6|7|X} . . . |
#H{0|1|2|3|4|5|6|7|8|9|A|B|C|D|E|F|X} . . . |
{0|1|2|3|4|5|6|7|8|9} . . . }"
```

Example

```
OUTPUT XXX;":MACHINE1:TLIST:DATA? 512, 'RAS'"
```

LINE

Command

```
:MACHine{1|2}:TLISt:LINE <line_num_mid_screen>
```

The LINE command allows you to scroll the timing analyzer listing vertically. The command specifies the state line number relative to the trigger that the analyzer highlights at the center of the screen.

`<line_num_mid_screen>` integer from −8191 to +8191

Example

```
OUTPUT XXX;":MACHINE1:TLIST:LINE 0"
```

Query

```
:MACHine{1|2}:TLISt:LINE?
```

The LINE query returns the line number for the state currently in the box at the center of the screen.

Returned Format

```
[:MACHine{1|2}:TLISt:LINE] <line_num_mid_screen><NL>
```

Example

```
OUTPUT XXX;":MACHINE1:TLIST:LINE?"
```

MMODe

Command

```
:MACHine{1|2}:TLISt:MMODe <marker_mode>
```

The MMODe command (Marker Mode) selects the mode controlling the marker movement and the display of marker readouts. When PATTern is selected, the markers will be placed on patterns. When TIME is selected, the markers move on time between stored states. When MSTats is selected, the markers are placed on patterns, but the readouts will be time statistics.

<marker_mode>

```
{OFF|PATTern|TIME|MSTats}
```

Example

```
OUTPUT XXX;":MACHINE1:TLIST:MMODE TIME"
```

Query

```
:MACHine{1|2}:TLISt:MMODe?
```

The MMODe query returns the current marker mode selected.

Returned Format

```
[:MACHine{1|2}:TLISt:MMODe] <marker_mode><NL>
```

Example

```
OUTPUT XXX;":MACHINE1:TLIST:MMODE?"
```

OCONdition

Command

```
:MACHine{1|2}:TLISt:OCONdition {ENTering|EXITing}
```

The OCONdition command specifies where the O marker is placed. The O marker can be placed on the entry or exit point of the OPATtern when in the PATTern marker mode.

Example

```
OUTPUT XXX; ":MACHINE1:TLIST:OCONDITION ENTERING"
```

Query

```
:MACHine{1|2}:TLISt:OCONdition?
```

The OCONdition query returns the current setting.

Returned Format

```
[:MACHine{1|2}:TLISt:OCONdition] {ENTering|EXITing}<NL>
```

Example

```
OUTPUT XXX;":MACHINE1:TLIST:OCONDITION?"
```

OPATtern

Command

```
:MACHine{1|2}:TLISt:OPATtern
<label_name>,<label_pattern>
```

The OPATtern command allows you to construct a pattern recognizer term for the O Marker which is then used with the OSEarch criteria when moving the marker on patterns. Since this command deals with only one label at a time, a complete specification could require several iterations.

When the value of a pattern is expressed in binary, it represents the bit values for the label inside the pattern recognizer term. In whatever base is used, the value must be between 0 and $2^{32} - 1$, since a label may not have more than 32 bits. Because the <label_pattern> parameter may contain don't cares, it is handled as a string of characters rather than a number.

`<label_name>` string of up to 6 alphanumeric characters

`<label_pattern>`

```
"{#B{0|1|X} . . . |
#Q{0|1|2|3|4|5|6|7|X} . . . |
#H{0|1|2|3|4|5|6|7|8|9|A|B|C|D|E|F|X} . . . |
{0|1|2|3|4|5|6|7|8|9} . . . }"
```

Examples

```
OUTPUT XXX;":MACHINE1:TLIST:OPATTERN 'DATA','255' "
OUTPUT XXX;":MACHINE1:TLIST:OPATTERN 'ABC','#BXXXX1101' "
```

Query

```
:MACHine{1|2}:TLISt:OPATtern? <label_name>
```

The OPATtern query returns the pattern specification for a given label name.

Returned Format

```
[:MACHine{1|2}:TLISt:OPATtern] <label_name>,<label_pattern><NL>
```

Example

```
OUTPUT XXX;":MACHINE1:TLIST:OPATTERN? 'A'"
```

OSEarch

Command

```
:MACHine{1|2}:TLISt:OSEarch <occurrence>,<origin>
```

The OSEarch command defines the search criteria for the O marker, which is then used with associated OPATtern recognizer specification when moving the markers on patterns. The origin parameter tells the marker to begin a search with the trigger, the start of data, or with the X marker. The actual occurrence the marker searches for is determined by the occurrence parameter of the OSEarch recognizer specification, relative to the origin. An occurrence of 0 places the marker on the selected origin. With a negative occurrence, the marker searches before the origin. With a positive occurrence, the marker searches after the origin.

`<occurrence>` integer from −8191 to +8191

`<origin>` `{TRIGger|STARt|XMARker}`

Example

```
OUTPUT XXX;":MACHINE1:TLIST:OSEARCH +10,TRIGGER"
```

Query

```
:MACHine{1|2}:TLISt:OSEarch?
```

The OSEarch query returns the search criteria for the O marker.

Returned Format

```
[:MACHine{1|2}:TLISt:OSEarch] <occurrence>,<origin><NL>
```

Example

```
OUTPUT XXX;":MACHINE1:TLIST:OSEARCH?"
```

OSTate

Query

```
:MACHine{1|2}:TLISt:OSTate?
```

The OSTate query returns the line number in the listing where the O marker resides (−8191 to +8191). If data is not valid, the query returns 32767.

Returned Format

```
[:MACHine{1|2}:TLISt:OSTate] <state_num><NL>
```

`<state_num>` an integer from −8191 to +8191, or 32767

Example

```
OUTPUT XXX;":MACHINE1:TLIST:OSTATE?"
```

OTAG

Command

```
:MACHine{1|2}:TLISt:OTAG <time_value>
```

The OTAG command specifies the tag value on which the O Marker should be placed. The tag value is time. If the data is not valid tagged data, no action is performed.

`<time_value>` real number

Example

```
:OUTPUT XXX;":MACHINE1:TLIST:OTAG 40.0E-6"
```

Query

```
:MACHine{1|2}:TLISt:OTAG?
```

The OTAG query returns the O Marker position in time regardless of whether the marker was positioned in time or through a pattern search. If data is not valid, the query returns 9.9E37 for time tagging, or returns 32767 for state tagging.

Returned Format

```
[:MACHine{1|2}:TLISt:OTAG] <time_value><NL>
```

Example

```
OUTPUT XXX;":MACHINE1:TLIST:OTAG?"
```

REMove

Command

```
:MACHine{1|2}:TLISt:REMove
```

The REMove command removes all labels, except the leftmost label, from the listing menu.

Example

```
OUTPUT XXX;":MACHINE1:TLIST:REMOVE"
```

RUNTil

Command

```
:MACHine{1|2}:TLISt:RUNTil <run_until_spec>
```

The RUNTil (run until) command allows you to define a stop condition when the trace mode is repetitive. Specifying OFF causes the analyzer to make runs until either the display's STOP field is touched, or, until the STOP command is issued.

There are four conditions based on the time between the X and O markers as follows:

- The difference is less than (LT) some value.
- The difference is greater than (GT) some value.
- The difference is inside some range (INRange).
- The difference is outside some range (OUTRange).

End points for the INRange and OUTRange should be at least 8 ns apart since this is the minimum time resolution of the time tag counter.

`<run_until_spec>` `{OFF|LT,<value>|GT,<value>|INRange,<value>,<value>|OUTRange,<value>,<value>}`

`<value>` real number from −9E9 to +9E9

Example

```
OUTPUT XXX;":MACHINE1:TLIST:RUNTIL GT,800.0E-6"
```

Query

```
:MACHine{1|2}:TLISt:RUNTil?
```

The RUNTil query returns the current stop criteria.

Returned Format

```
[:MACHine{1|2}:TLISt:RUNTil] <run_until_spec><NL>
```

Example

```
OUTPUT XXX;":MACHINE1:TLIST:RUNTIL?"
```

TAVerage

Query `:MACHine{1|2}:TLISt:TAVerage?`

The TAVerage query returns the value of the average time between the X and O Markers. If the number of valid runs is zero, the query returns 9.9E37. Valid runs are those where the pattern search for both the X and O markers was successful, resulting in valid delta-time measurements.

Returned Format `[:MACHine{1|2}:TLISt:TAVerage] <time_value><NL>`

`<time_value>` real number

Example

```
OUTPUT XXX;":MACHINE1:TLIST:TAVERAGE?"
```

TMAXimum

Query `:MACHine{1|2}:TLISt:TMAXimum?`

The TMAXimum query returns the value of the maximum time between the X and O Markers. If data is not valid, the query returns 9.9E37.

Returned Format `[:MACHine{1|2}:TLISt:TMAXimum] <time_value><NL>`

`<time_value>` real number

Example

```
OUTPUT XXX;":MACHINE1:TLIST:TMAXIMUM?"
```

TMINimum

Query

```
:MACHine{1|2}:TLISt:TMINimum?
```

The TMINimum query returns the value of the minimum time between the X and O Markers. If data is not valid, the query returns 9.9E37.

Returned Format

```
[:MACHine{1|2}:TLISt:TMINimum] <time_value><NL>
```

`<time_value>` real number

Example

```
OUTPUT XXX;":MACHINE1:TLIST:TMINIMUM?"
```

VRUNs

Query

```
:MACHine{1|2}:TLISt:VRUNs?
```

The VRUNs query returns the number of valid runs and total number of runs made. Valid runs are those where the pattern search for both the X and O markers was successful resulting in valid delta time measurements.

Returned Format

```
[:MACHine{1|2}:TLISt:VRUNs] <valid_runs>,<total_runs><NL>
```

`<valid_runs>` zero or positive integer

`<total_runs>` zero or positive integer

Example

```
OUTPUT XXX;":MACHINE1:TLIST:VRUNS?"
```

XCONdition

Command

```
:MACHine{1|2}:TLISt:XCONdition {ENTering|EXITing}
```

The XCONdition command specifies where the X marker is placed. The X marker can be placed on the entry or exit point of the XPATtern when in the PATTern marker mode.

Example

```
OUTPUT XXX; ":MACHINE1:TLIST:XCONDITION ENTERING"
```

Query

```
:MACHine{1|2}:TLISt:XCONdition?
```

The XCONdition query returns the current setting.

Returned Format

```
[:MACHine{1|2}:TLISt:XCONdition] {ENTering|EXITing}<NL>
```

Example

```
OUTPUT XXX;":MACHINE1:TLIST:XCONDITION?"
```

XOTag

Query

```
:MACHine{1|2}:TLISt:XOTag?
```

The XOTag query returns the time from the X to O markers. If there is no data in the time mode the query returns 9.9E37.

Returned Format

```
[:MACHine{1|2}:TLISt:XOTag] <XO_time><NL>
```

<XO_time> real number

Example

```
OUTPUT XXX;":MACHINE1:TLIST:XOTAG?"
```

XOTime

Query

```
:MACHine{1|2}:TLISt:XOTime?
```

The XOTime query returns the time from the X to O markers. If there is no data in the time mode the query returns 9.9E37.

Returned Format

```
[:MACHine{1|2}:TLISt:XOTime] <XO_time><NL>
```

`<XO_time>` real number

Example

```
OUTPUT XXX;":MACHINE1:TLIST:XOTIME?"
```

XPATtern

Command

```
:MACHine{1|2}:TLISt:XPATtern <label_name>,
<label_pattern>
```

The XPATtern command allows you to construct a pattern recognizer term for the X Marker which is then used with the XSEarch criteria when moving the marker on patterns. Since this command deals with only one label at a time, a complete specification could require several iterations.

When the value of a pattern is expressed in binary, it represents the bit values for the label inside the pattern recognizer term. In whatever base is used, the value must be between 0 and $2^{32} - 1$, since a label may not have more than 32 bits. Because the `<label_pattern>` parameter may contain don't cares, it is handled as a string of characters rather than a number.

`<label_name>` string of up to 6 alphanumeric characters

`<label_pattern>`

```
"{#B{0|1|X} . . . |
#Q{0|1|2|3|4|5|6|7|X} . . . |
#H{0|1|2|3|4|5|6|7|8|9|A|B|C|D|E|F|X} . . . |
{0|1|2|3|4|5|6|7|8|9} . . . }"
```

Examples

```
OUTPUT XXX;":MACHINE1:TLIST:XPATTERN 'DATA','255' "
OUTPUT XXX;":MACHINE1:TLIST:XPATTERN 'ABC','#BXXXX1101' "
```

Query

```
:MACHine{1|2}:TLISt:XPATtern? <label_name>
```

The XPATtern query returns the pattern specification for a given label name.

Returned Format

```
[:MACHine{1|2}:TLISt:XPATtern] <label_name>,<label_pattern><NL>
```

Example

```
OUTPUT XXX;":MACHINE1:TLIST:XPATTERN? 'A'"
```

XSEarch

Command

```
:MACHine{1|2}:TLISt:XSEarch <occurrence>,<origin>
```

The XSEarch command defines the search criteria for the X Marker, which is then with associated XPATtern recognizer specification when moving the markers on patterns. The origin parameter tells the marker to begin a search with the trigger or with the start of data. The occurrence parameter determines which occurrence of the XPATtern recognizer specification, relative to the origin, the marker actually searches for. An occurrence of 0 places a marker on the selected origin.

`<occurrence>` integer from −8191 to +8191

`<origin>` `{TRIGger|STARt}`

Example

```
OUTPUT XXX;":MACHINE1:TLIST:XSEARCH +10,TRIGGER"
```

Query `:MACHine{1|2}:TLISt:XSEarch?`

The XSEarch query returns the search criteria for the X marker.

Returned Format `[:MACHine{1|2}:TLISt:XSEarch] <occurrence>,<origin><NL>`

Example

```
OUTPUT XXX;":MACHINE1:TLIST:XSEARCH?"
```

XSTate

Query `:MACHine{1|2}:TLISt:XSTate?`

The XSTate query returns the line number in the listing where the X marker resides (−8191 to +8191). If data is not valid, the query returns 32767.

Returned Format `[:MACHine{1|2}:TLISt:XSTate] <state_num><NL>`

`<state_num>` an integer from −8191 to +8191, or 32767

Example

```
OUTPUT XXX;":MACHINE1:TLIST:XSTATE?"
```

XTAG

Command

```
:MACHine{1|2}:TLISt:XTAG <time_value>
```

The XTAG command specifies the tag value on which the X Marker should be placed. The tag value is time. If the data is not valid tagged data, no action is performed.

`<time_value>` real number

Example

```
OUTPUT XXX;":MACHINE1:TLIST:XTAG 40.0E-6"
```

Query

```
:MACHine{1|2}:TLISt:XTAG?
```

The XTAG query returns the X Marker position in time regardless of whether the marker was positioned in time or through a pattern search. If data is not valid tagged data, the query returns 9.9E37.

Returned Format

```
[:MACHine{1|2}:TLISt:XTAG] <time_value><NL>
```

Example

```
OUTPUT XXX;":MACHINE1:TLIST:XTAG?"
```

25

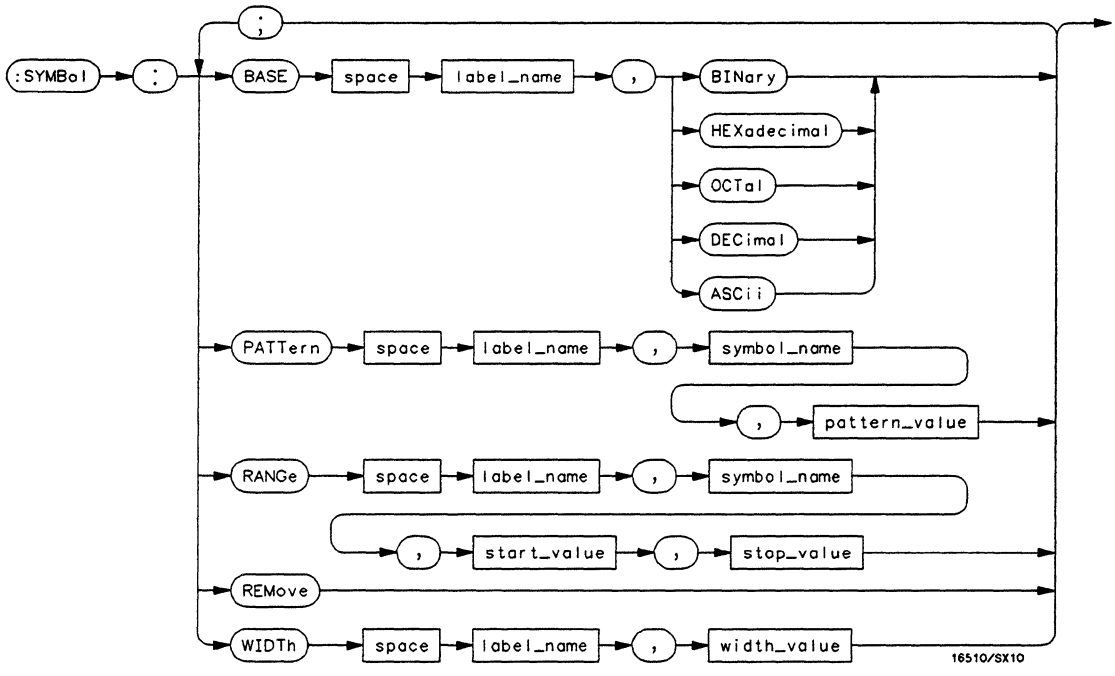
SYMBol Subsystem

Introduction

The SYMBol subsystem contains the commands that allow you to define symbols on the controller and download them to the HP 1660-series logic analyzers. The commands in this subsystem are:

- BASE
- PATTern
- RANGe
- REMove
- WIDTh

Figure 25-1



SYMBol Subsystem Syntax Diagram

Table 25-1 **SYMBol Parameter Values**

Parameter	Values
label_name	string of up to 6 alphanumeric characters
symbol_name	string of up to 16 alphanumeric characters
pattern_value	"{#B{0|1|X}...| #Q{0|1|2|3|4|5|6|7|X}...| #H{0|1|2|3|4|5|6|7|8|9|A|B|C|D|E|F|X}...| {0|1|2|3|4|5|6|7|8|9}...}"
start_value	"{#B{0|1}...| #Q{0|1|2|3|4|5|6|7}...| #H{0|1|2|3|4|5|6|7|8|9|A|B|C|D|E|F}...| {0|1|2|3|4|5|6|7|8|9}...}"
stop_value	"{#B{0|1}...| #Q{0|1|2|3|4|5|6|7}...| #H{0|1|2|3|4|5|6|7|8|9|A|B|C|D|E|F}...| {0|1|2|3|4|5|6|7|8|9}...}"
width_value	integer from 1 to 16

SYMBol

Selector

```
:MACHine{1|2}:SYMBol
```

The SYMBol selector is used as a part of a compound header to access the commands used to create symbols. It always follows the MACHine selector because it selects a branch directly below the MACHine level in the command tree.

Example

```
OUTPUT XXX;":MACHINE1:SYMBOL:BASE 'DATA', BINARY"
```

BASE

Command `:MACHine{1|2}:SYMBol:BASE <label_name>,<base_value>`

The BASE command sets the base in which symbols for the specified label will be displayed in the symbol menu. It also specifies the base in which the symbol offsets are displayed when symbols are used.

> BINary is not available for labels with more than 20 bits assigned. In this case the base will default to HEXadecimal.

`<label_name>` string of up to 6 alphanumeric characters

`<base_value>` `{BINary|HEXadecimal|OCTal|DECimal|ASCii}`

Example

```
OUTPUT XXX;":MACHINE1:SYMBOL:BASE 'DATA',HEXADECIMAL"
```

PATTern

Command

```
:MACHine{1|2}:SYMBol:PATTern <label_name>,
<symbol_name>,<pattern_value>
```

The PATTern command allows you to create a pattern symbol for the specified label. Because don't cares (X) are allowed in the pattern value, it must always be expressed as a string. You may still use different bases, though don't cares cannot be used in a decimal number.

`<label_name>` string of up to 6 alphanumeric characters

`<symbol_name>` string of up to 16 alphanumeric characters

`<pattern_value>`

```
"{#B{0|1|X} . . . |
#Q{0|1|2|3|4|5|6|7|X} . . . |
#H{0|1|2|3|4|5|6|7|8|9|A|B|C|D|E|F|X} . . . |
{0|1|2|3|4|5|6|7|8|9} . . . }"
```

Example

```
OUTPUT XXX;":MACHINE1:SYMBOL:PATTERN 'STAT', 'MEM_RD','#H01XX'"
```

RANGe

Command

```
:MACHine{1|2}:SYMBol:RANGe <label_name>,
<symbol_name>,<start_value>,<stop_value>
```

The RANGe command allows you to create a range symbol containing a start value and a stop value for the specified label. The values may be in binary (`#B`), octal (`#Q`), hexadecimal (`#H`) or decimal (default). You can not use don't cares in any base.

<label_name>	string of up to 6 alphanumeric characters
<symbol_name>	string of up to 16 alphanumeric characters
<start_value>	"{#B{0|1} . . . | #Q{0|1|2|3|4|5|6|7} . . . | #H{0|1|2|3|4|5|6|7|8|9|A|B|C|D|E|F} . . . | {0|1|2|3|4|5|6|7|8|9} . . . }"
<stop_value>	"{#B{0|1} . . . | #Q{0|1|2|3|4|5|6|7} . . . | #H{0|1|2|3|4|5|6|7|8|9|A|B|C|D|E|F} . . . | {0|1|2|3|4|5|6|7|8|9} . . . }"

Example

```
OUTPUT XXX;":MACHINE1:SYMBOL:RANGE 'STAT',
'IO_ACC','0','#H000F'"
```

REMove

Command

```
:MACHine{1|2}:SYMBol:REMove
```

The REMove command deletes all symbols from a specified machine.

Example

```
OUTPUT XXX;":MACHINE1:SYMBOL:REMOVE"
```

WIDTh

Command

```
:MACHine{1|2}:SYMBol:WIDTh <label_name>,
<width_value>
```

The WIDTh command specifies the width (number of characters) in which the symbol names will be displayed when symbols are used.

> The WIDTh command does not affect the displayed length of the symbol offset value.

`<label_name>` string of up to 6 alphanumeric characters

`<width_value>` integer from 1 to 16

Example

```
OUTPUT XXX;":MACHINE1:SYMBOL:WIDTH 'DATA',9 "
```

26

DATA and SETup Commands

Introduction

The DATA and SETup commands are SYSTem commands that allow you to send and receive block data between the HP 1660 series and a controller. Use the DATA instruction to transfer acquired timing and state data, and the SETup instruction to transfer instrument configuration data. This is useful for:

- Re-loading to the logic analyzer
- Processing data later
- Processing data in the controller

This chapter explains how to use these commands.

The format and length of block data depends on the instruction being used, the configuration of the instrument, and the amount of acquired data. The length of the data block can be up to 409,760 bytes in the HP 1660A.

The SYSTem:DATA section describes each part of the block data as it will appear when used by the DATA instruction. The beginning byte number, the length in bytes, and a short description is given for each part of the block data. This is intended to be used primarily for processing of data in the controller.

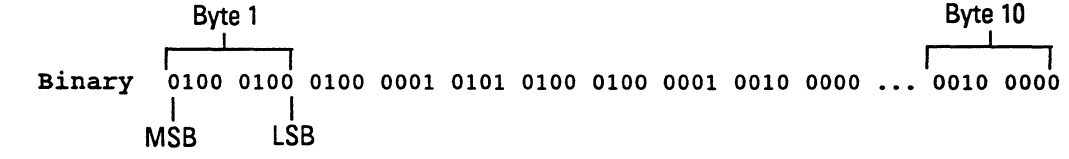
Do not change the block data in the controller if you intend to send the block data back into the logic analyzer for later processing. Changes made to the block data in the controller could have unpredictable results when sent back to the logic analyzer.

Data Format

To understand the format of the data within the block data, there are four important things to keep in mind.

- Data is sent to the controller in binary form.
- Each byte, as described in this chapter, contains 8 bits.
- The first bit of each byte is the MSB (most significant bit).
- Byte descriptions are printed in binary, decimal, or ASCII depending on how the data is described.

For example, the first ten bytes that describe the section name contain a total of 80 bits as follows:




```
Decimal  68 65 84 65 32 32 32 32 32 32
  ASCII  DATA space space space space space space
```

:SYSTem:DATA

Command

```
:SYSTem:DATA <block_data>
```

The SYSTem:DATA command transmits the acquisition memory data from the controller to the HP 1660-series logic analyzer.

The block data consists of a variable number of bytes containing information captured by the acquisition chips. The information will be in one of three formats, depending on the type of data captured. The three formats are glitch, transitional, conventional timing or state. Each format is described in the "Acquisition Data Description" section later in this chapter. Since no parameter checking is performed, out-of-range values could cause instrument lockup; therefore, care should be taken when transferring the data string into the logic analyzer.

The <block_data> parameter can be broken down into a <block_length_specifier> and a variable number of <section>'s.

The <block_length_specifier> always takes the form #8DDDDDDDD. Each D represents a digit (ASCII characters "0" through "9"). The value of the eight digits represents the total length of the block (all sections). For example, if the total length of the block is 14522 bytes, the block length specifier would be "#800014522".

Each <section> consists of a <section header> and <section data>. The <section data> format varies for each section. For the DATA instruction, there is only one <section>, which is composed of a data preamble followed by the acquisition data. This section has a variable number of bytes depending on configuration and amount of acquired data.

`<block_data>`	`<block_length_specifier><section>`
`<block_length_specifier>`	`#8<length>`
`<length>`	The total length of all sections in byte format (must be represented with 8 digits)
`<section>`	`<section header><section data>`
`<section_header>`	16 bytes, described in "Section Header Description," on page 26-6.
`<section_data>`	Format depends on the specific section.

Example

```
OUTPUT XXX;":SYSTEM:DATA" <block_data>
```

> The total length of a section is 16 (for the section header) plus the length of the section data. So when calculating the value for `<length>`, don't forget to include the length of the section headers.

Query

```
:SYSTem:DATA?
```

The SYSTem:DATA query returns the block data to the controller. The data sent by the SYSTem:DATA query reflect the configuration of the machines when the last run was performed. Any changes made since then through either front-panel operations or programming commands do not affect the stored configuration.

Returned Format

```
[:SYSTem:DATA] <block_data><NL>
```

Example

See "Transferring the logic analyzer acquired data" on page 27-17 in chapter 27, "Programming Examples" for an example.

Section Header Description

The section header uses bytes 1 through 16 (this manual begins counting at 1; there is no byte 0). The 16 bytes of the section header are as follows:

Byte Position	
1	10 bytes - Section name ("DATA space space space space space space" in ASCII for the DATA instruction).
11	1 byte - Reserved
12	1 byte - Module ID (0010 0000 binary or 32 decimal for the HP 1660 series)
13	4 bytes - Length of section in number of bytes that, when converted to decimal, specifies the number of bytes contained in the section.

Section Data

For the SYSTem:DATA command, the <section data> parameter consists of two parts: the data preamble and the acquisition data. These are described in the following two sections.

Data Preamble Description

The block data is organized as 160 bytes of preamble information, followed by a variable number of bytes of data. The preamble gives information for each analyzer describing the amount and type of data captured, where the trace point occurred in the data, which pods are assigned to which analyzer, and other information. The values stored in the preamble represent the captured data currently stored in this structure and not the current analyzer configuration. For example, the mode of the data (bytes 21 and 49) may be STATE with tagging, while the current setup of the analyzer is TIMING.

The preamble (bytes 17 through 176) consists of the following 160 bytes:

17	2 bytes - Instrument ID (always 1660 decimal for HP 1660 series)
19	1 byte - Revision Code
20	1 byte - number of acquisition chips used in last acquisition

The next 40 bytes are for Analyzer 1 Data Information.

`Byte Position`

21 1 byte - Machine data mode, one of the following decimal values:
- −1 = off
- 0 = state data without tags
- 1 = state data with each chip assigned to a machine (2kB memory) and either time or state tags
- 2 = state data with unassigned pod used to store tag data (4kB memory)
- 8 = state data at half channel (8kB memory with no tags)
- 10 = conventional timing data at full channel
- 11 = transitional timing data at full channel
- 12 = glitch timing data
- 13 = conventional timing data at half channel
- 14 = transitional timing data at half channel

22 1 byte - Unused.

23 2 bytes - List of pods in this analyzer, where a binary 1 indicates that the corresponding pod is assigned to this analyzer

bit 15	bit 14	bit 13	bit 12	bit 11	bit 10	bit 9	bit 8
unused	unused	always 1	unused	unused	unused	unused	Pod 8[1]
bit 7	**bit 6**	**bit 5**	**bit 4**	**bit 3**	**bit 2**	**bit 1**	**bit 0**
Pod 7[1]	Pod 6[2]	Pod 5[2]	Pod 4[3]	Pod 3[3]	Pod 2	Pod 1	unused

1 – also unused in the HP 1661A, HP 1662A, and HP 1663A
2 – also unused in the HP 1662A and HP 1663A
3 – also unused in the HP 1663A

Example

`xx10 0000 0001 111x` indicates pods 1 through 4 are assigned to this analyzer (x = unused bit).

25 1 byte - This byte returns which chip is used to store the time or state tags when an unassigned pod is available to store tag data. This chip is available in state data mode with an unassigned pod and state or time tags on. Byte 21 = 2 in this mode.

`Byte Position`

`26` 1 byte - Master chip for this analyzer. This decimal value returns which chip's time tag data is valid in a non-transitional mode; for example, state with time tags.

5 - pods 1 and 2	2 - pods 7 and 8[3]
4 - pods 3 and 4[1]	1 - unused
3 - pods 5 and 6[2]	0 - unused
	–1 - no chip

1 – also unused in the HP 1663A
2 – also unused in the HP 1662A and HP 1663A
3 – also unused in the HP 1661A, HP 1662A, and HP 1663A

`27` 6 bytes - Unused

`33` 8 bytes - A decimal integer representing sample period in picoseconds (timing only).

Example The following 64 bits in binary would equal 8,000 picoseconds or, 8 nanoseconds:

```
00000000 00000000 00000000 00000000 00000000 00000000 00011111 01000000
```

`41` 8 bytes - Unused

`49` 1 byte - Tag type for state only in one of the following decimal values:
0 = off
1 = time tags
2 = state tags

`50` 1 byte - Unused

`51` 8 bytes - A decimal integer representing the time offset in picoseconds from when this analyzer is triggered and when this analyzer provides an output trigger to the IMB or port out. The value for one analyzer is always zero and the value for the other analyzer is the time between the triggers of the two analyzers.

`59` 2 bytes - Unused

Byte Position

61 40 bytes - The next 40 bytes are for Analyzer 2 Data Information. They are organized in the same manner as Analyzer 1 above, but they occupy bytes 61 through 100.

101 26 bytes - Number of valid rows of data (starting at byte 177) for each pod. The 26 bytes of this group are organized as follows:
Bytes 1 and 2 - Unused
Bytes 3 and 4 - Unused.
Bytes 5 and 6 - Unused.
Bytes 7 and 8 - Unused.
Bytes 9 and 10 - Unused.
Bytes 11 and 12 contain the number of valid rows of data for pod 8 of the HP 1660A only. Unused in the other HP 1660-series logic analyzers.
Bytes 13 and 14 contain the number of valid rows of data for pod 7 of the HP 1660A only. Unused in the other HP 1660-series logic analyzers
Bytes 15 and 16 contain the number of valid rows of data for pod 6 of the HP 1660A and HP 1661A only.
Bytes 17 and 18 contain the number of valid rows of data for pod 5 of the HP 1660A and HP 1661A only.
Bytes 19 and 20 contain the number of valid rows of data for pod 4 of the HP 1660A, HP 1661A, and HP 1662A only.
Bytes 21 and 22 contain the number of valid rows of data for pod 3 of the HP 1660A, HP 1661A, and HP 1662A only.
Bytes 23 and 24 contain the number of valid rows of data for pod 2 of all models of the HP1660-series logic analyzers.
Bytes 25 and 26 contain the number of valid rows of data for pod 1 of all models of the HP1660-series logic analyzers.

Byte Position	
127	26 bytes - Row of data containing the trigger point. This byte group is organized in the same way as the data rows (starting at byte 101 above). These binary numbers are base zero numbers which start from the first sample stored for a specific pod. For example, if bytes 151 and 152 contained a binary number with a decimal equivalent of +1018, the data row having the trigger is the 1018th data row on pod 1. There are 1018 rows of pre-trigger data as shown below. row 0 row 1 . . . row 1017 row 1018 – trigger row
153	24 bytes - Unused

Acquisition Data Description

The acquisition data section consists of a variable number of bytes depending on which logic analyzer you are using, the acquisition mode and the tag setting (time, state, or off). The data is grouped in 18-byte rows for the HP 1660A, in 14-byte rows for the HP 1661A, in 10-byte rows for the HP 1662A, and in 6-byte rows for the HP 1663A.

The number of rows for each pod is stored in byte positions 101 through 126. The number of bytes in each row can be determined by the value stored in byte position 20 which contains the number of acquisition chips in the instrument. For example, if the value in byte position 20 is 4, the instrument is an HP 1660A. Values 3, 2, and 1 represent the HP 1661A, 1662A, and 1663A respectively.

`Byte Position`

	clock lines	Pod 8[1]	Pod 7[1]	pod 6[2]	pod 5[2]	pod 4[3]	pod 3[3]	pod 2	pod 1[4]
177	2 bytes	2 bytes	2 bytes	2 bytes	2 bytes	2 bytes	2 bytes	2 bytes	2 bytes
195	2 bytes	2 bytes	2 bytes	2 bytes	2 bytes	2 bytes	2 bytes	2 bytes	2 bytes
•	•	•	•	•	•	•	•	•	•
•	•	•	•	•	•	•	•	•	•
•	•	•	•	•	•	•	•	•	•
(x)	2 bytes	2 bytes	2 bytes	2 bytes	2 bytes	2 bytes	2 bytes	2 bytes	2 bytes

1 – unused in the HP 1661A, HP 1662A, and HP 1663A
2 – also unused in the HP 1662A and HP 1663 A
3 – also unused in the HP 1663A
4 – The headings are not a part of the returned data.

Row (x) is the highest number of valid rows specified by the bytes in byte positions 101 through 126 in all modes and when neither analyzer is in glitch mode. In the glitch mode, row (x) is the larger of:

1. The highest number of valid rows specified by the bytes in byte positions 101 through 126; or,
2. 2048 + the highest number of valid rows for the pods assigned to the timing analyzer, when one or more glitches are detected.

The clock-line bytes for the HP 1660A, which also includes 2 additional data lines (D), are organized as follows:

```
xxxx xxPN xxDD MLKJ
```

The clock-line bytes for the HP 1661A and HP 1662A are organized as follows:

```
xxxx xxxx xxxx MLKJ
```

The clock-line bytes for the HP 1663A are organized as follows:

```
xxxx xxxx xxxx xxKJ
```

Time Tag Data Description

The time tag data starts at the end of the acquired data. Each data row has an 8-byte time tag for each chip (2-pod set). The starting location of the time tag data is immediately after the last row of valid data (maximum data byte + 1). If an analyzer is in a non-transitional mode, the master chip (byte 26) is the only chip with valid time-tag data. The time tag data is a decimal integer representing time in picoseconds for both timing and state time tags. For state tags in the state analyzer, tag data is a decimal integer representing the number of states.

Time Tag Block (for the HP 1660A)

Byte 1 through 8 (64 bits starting with the MSB) - First sample tag for pods 1 and 2.

Byte 9 through 16 (64 bits starting with the MSB) - Second sample tag for pods 1 and 2.

.
.
.

Byte (w) through (w + 7) (64 bits starting with the MSB) - Last sample tag for pods 1 and 2.

Byte (w + 8) through (w + 15) (64 bits starting with the MSB) - First sample tag for pods 3 and 4.

Byte (w + 16) through (w + 23) (64 bits starting with the MSB) - Second sample tag for pods 3 and 4.

.
.
.

Byte (x) through (x+ 7) (64 bits starting with the MSB) - Last sample tag for pods 3 and 4.

Byte (x + 8) through (x + 15) (64 bits starting with the MSB) - First sample tag for pods 5 and 6.

Byte (x + 16) through (x + 23) (64 bits starting with the MSB) - Second sample tag for pods 5 and 6.

.
.
.

Byte (y) through (y+ 7) (64 bits starting with the MSB) - Last sample tag for pods 5 and 6.

Byte (y + 8) through (y + 15) (64 bits starting with the MSB) - First sample tag for pods 7 and 8.

Byte (y + 16) through (y + 23) (64 bits starting with the MSB) - Second sample tag for pods 7 and 8.

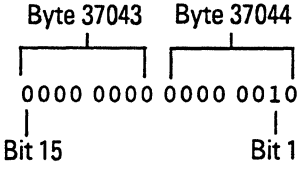
.
.
.

Byte (z) through (z+ 7) (64 bits starting with the MSB) - Last sample tag for pods 7 and 8.

Glitch Data Description

In the glitch mode, each pod has two bytes assigned to indicate where glitches occur in the acquired data. For each row of acquired data there will be a corresponding row of glitch data. The glitch data is organized in the same way as the acquired data. The glitch data is grouped in 18-byte rows for the HP 1660A. The number of rows is stored in byte positions 101 through 126. The starting byte of the glitch data is an absolute starting point regardless of the number of rows of acquired data.

A binary 1 in the glitch data indicates a glitch was detected. For example, if a glitch occurred on bit 1 of pod 8 in data row 1 of an HP 1660A, bytes 37043 and 37044 would contain:



`Byte Position`

	clock lines	Pod 8[1]	Pod 7[1]	pod 6[2]	pod 5[2]	pod 4[3]	pod 3[3]	pod 2	pod 1[4]
37041	2 bytes	2 bytes	2 bytes	2 bytes	2 bytes	2 bytes	2 bytes	2 bytes	2 bytes
37059	2 bytes	2 bytes	2 bytes	2 bytes	2 bytes	2 bytes	2 bytes	2 bytes	2 bytes
•	•	•	•	•	•	•	•	•	•
•	•	•	•	•	•	•	•	•	•
•	•	•	•	•	•	•	•	•	•
(x)	2 bytes	2 bytes	2 bytes	2 bytes	2 bytes	2 bytes	2 bytes	2 bytes	2 bytes

1 – unused in the HP 1661A, HP 1662A, and HP 1663A
2 – also unused in the HP 1662A and HP 1663 A
3 – also unused in the HP 1663A
4 – The headings are not a part of the returned data.

SYSTem:SETup

Command

```
:SYStem:SETup <block_data>
```

The SYStem:SETup command configures the logic analyzer module as defined by the block data sent by the controller. The length of the configuration data block can be up to 350,784 bytes in the HP 1660A.

There are four data sections which are always returned. These are the strings which would be included in the section header:

```
"CONFIG     "
"DISPLAY1   "
"BIG_ATTRIB"
"RTC_INFO   "
```

Additionally, the following sections may also be included, depending on what's available:

```
"SYMBOLS A "
"SYMBOLS B "
"INVASM A  "
"INVASM B  "
"COMPARE   "
```

With the exception of the RTC_INFO section, the block data is not described. However, the RTC_INFO section contains the real-time clock time of the acquired data in the data block. This time information can be meaningful to some measurements.

`<block_data>`	`<block_length_specifier><section>`
`<block_length_specifier>`	`#8<length>`
`<length>`	The total length of all sections in byte format (must be represented with 8 digits)
`<section>`	`<section_header><section_data>[<section_data>...]`
`<section_header>`	16 bytes in the following format: 10 bytes for the section name 1 byte reserved 1 byte for the module ID code (32 for the HP 1660-series logic analyzer) 4 bytes for the length of section data in number of bytes that, when converted to decimal, specifies the number of bytes contained in the section. The RTC_INFO section is described in the "RTC_INFO Section Description."
`<section_data>`	Format depends on the section.

> The total length of a section is 16 (for the section header) plus the length of the section data. So when calculating the value for <length>, don't forget to include the length of the section headers.

Example

```
OUTPUT XXX;"SETUP" <block_data>
```

Query

```
:SYStem:SETup?
```

The SYStem:SETup query returns a block of data that contains the current configuration to the controller.

Returned Format

```
[:SYStem:SETup] <block_data><NL>
```

Example

See "Transferring the logic analyzer configuration" on page 27-14 in Chapter 27, "Programming Examples" for an example.

RTC_INFO Section Description

The RTC_INFO section contains the real time of the acquired data. Because the time of the acquired data is important to certain measurements, this section describes how to find the real-time clock data.

Because the number of sections in the SETup data block depends on the logic analyzer configuration, the RTC_INFO section will not always be in the same location within the block. Therefore, the section must be found by name. Once the section is found, you can find the time by using the description in the following section:

```
#8<block_length>...[<section_name><section_length>
<section_data>]...
```

`<block_length>`	Total length of all sections
`<section_name>`	10 bytes - Section name. `"RTC_INFO space space"`
`<section_length>`	4 bytes - Length of section. 8 bytes, decimal, for RTC_INFO section.
`<section_data>`	10 bytes - Contains the real-time clock data described as follows:

Byte Position	
1	1 byte - Year. A decimal integer that, when added to 1990, defines the year. For example, if this byte has a decimal value of 2, the year is 1992.
2	1 byte - Month. An integer from 1 to 12.
3	1 byte - Day. An integer from 1 to 31.
4	1 byte - Unused
5	1 byte - Hour. An integer from 1 to 23.
6	1 byte - Minute. An integer from 1 to 59.
7	1 byte - Second. An integer from 1 to 59.
8	1 byte - Unused.

27

Programming Examples

Introduction

This chapter contains short, usable, and tested program examples that cover the most asked for examples. The examples are written in HP Basic 6.0.

- Making a timing analyzer measurement
- Making a state analyzer measurement
- Making a state compare measurement
- Transferring logic analyzer configuration between the logic analyzer and the controller
- Transferring logic analyzer data between the logic analyzer and the controller
- Checking for measurement completion
- Sending queries to the logic analyzer
- Getting ASCII data with PRINt? All query
- Reading a disk catalog
- Printing to the disk using PRINT? ALL

Making a Timing analyzer measurement

This program sets up the logic analyzer to make a simple timing analyzer measurement. This example can be used with E2433-60004 Logic Analyzer Training board to acquire and display the output of the ripple counter. It can also be modified to make any timing analyzer measurement.

```
10    ! ****************** TIMING ANALYZER EXAMPLE ******************
20    !                 for the HP 1660A Logic Analyzer
30    !
40    ! ************************************************************
50    ! Select the module slot in which the logic analyzer is installed.
60    ! Always a 1 for the HP 1660-series logic analyzers.
70    !
80    OUTPUT 707;":SELECT 1"
90    !
100   ! ************************************************************
110   ! Name Machine 1 "TIMING," configure Machine 1 as a timing analyzer,
120   ! and assign pod 1 to Machine 1.
130   !
140   OUTPUT 707;":MACH1:NAME 'TIMING'"
150   OUTPUT 707;":MACH1:TYPE TIMING"
160   OUTPUT 707;":MACH1:ASSIGN 1"
170   !
180   ! ************************************************************
190   ! Make a label "COUNT," give the label a positive polarity, and
200   ! assign the lower 8 bits.
210   !
220   OUTPUT 707;":MACHINE1:TFORMAT:REMOVE ALL"
230   OUTPUT 707;":MACH1:TFORMAT:LABEL 'COUNT',POS,0,0,#B0000000011111111"
240   !
250   ! ************************************************************
260   ! Specify FF hex for resource term A, which is the default trigger term for
270   ! the timing analyzer.
280   !
290   OUTPUT 707;":MACH1:TTRACE:TERM A, 'COUNT', '#HFF'"
300   !
310   ! ************************************************************
320   ! Remove any previously inserted labels, insert the "COUNT"
330   ! label, change the seconds-per-division to 100 ns, and display the
340   ! waveform menu.
350   !
```

```
360   OUTPUT 707;":MACH1:TWAVEFORM:REMOVE"
370   OUTPUT 707;":MACH1:TWAVEFORM:INSERT 'COUNT', ALL"
380   OUTPUT 707;":MACH1:TWAVEFORM:RANGE 1E-6"
390   OUTPUT 707;":MENU 1,5"
400   !
410   ! ***************************************************************
420   ! Run the timing analyzer in single mode.
430   !
440   OUTPUT 707;":RMODE SINGLE"
450   OUTPUT 707;":START"
460   !
470   ! ***************************************************************
480   ! Set the marker mode (MMODE) to time so that time tags are available
490   ! for marker measurements.  Place the X-marker on 03 hex and the O-
500   ! marker on 07 hex.  Then tell the timing analyzer to find the first
510   ! occurrence of 03h after the trigger and the first occurrence of 07h
520   ! after the X-marker is found.
530   !
540   OUTPUT 707;":MACHINE1:TWAVEFORM:MMODE TIME"
550   !
560   OUTPUT 707;":MACHINE1:TWAVEFORM:XPATTERN 'COUNT','#H03'"
570   OUTPUT 707;":MACHINE1:TWAVEFORM:OPATTERN 'COUNT','#H07'"
580   !
590   OUTPUT 707;":MACHINE1:TWAVEFORM:XCONDITION ENTERING"
600   OUTPUT 707;":MACHINE1:TWAVEFORM:OCONDITION ENTERING"
610   !
620   OUTPUT 707;":MACHINE1:TWAVEFORM:XSEARCH +1, TRIGGER"
630   OUTPUT 707;":MACHINE1:TWAVEFORM:OSEARCH +1, XMARKER"
640   !
650   ! ***************************************************************
660   ! Turn the longform and headers on, dimension a string for the query
670   ! data, send the XOTIME query and print the string containing the
680   ! XOTIME query data.
690   !
700   OUTPUT 707;":SYSTEM:LONGFORM ON"
710   OUTPUT 707;":SYSTEM:HEADER ON"
720   !
730   DIM Mtime$[100]
740   OUTPUT 707;":MACHINE1:TWAVEFORM:XOTIME?"
750   ENTER 707;Mtime$
760   PRINT Mtime$
770   END
```

Making a State analyzer measurement

This state analyzer program selects the HP 1660-series logic analyzer, displays the configuration menu, defines a state machine, displays the state trigger menu, sets a state trigger for multilevel triggering. This program then starts a single acquisition measurement while checking for measurement completion.

This program is written in such a way you can run it with the HP E2433-60004 Logic Analyzer Training Board. This example is the same as the "Multilevel State Triggering" example in chapter 9 of the *HP E2433-90910 Logic Analyzer Training Guide.*

```
10    ! ******************** STATE ANALYZER EXAMPLE *************************
20    !                  for the HP 1660-series Logic Analyzers
30    !
40    ! ****************** SELECT THE LOGIC ANALYZER **********************
50    ! Select the module slot in which the logic analyzer is installed.
60    ! Always a 1 for the HP 1660-series logic analyzers.
70    !
80    OUTPUT 707;":SELECT 1"
90    !
100   ! ******************** CONFIGURE THE STATE ANALYZER **********************
110   ! Name Machine 1 "STATE," configure Machine 1 as a state analyzer, assign
120   ! pod 1 to Machine 1, and display System Configuration menu of the
130   ! logic analyzer.
140   !
150   OUTPUT 707;":MACHINE1:NAME 'STATE'"
160   OUTPUT 707;":MACHINE1:TYPE STATE"
170   OUTPUT 707;":MACHINE1:ASSIGN 1"
180   OUTPUT 707;":MENU 1,0"
190   !
200   ! ******************* SETUP THE FORMAT SPECIFICATION ********************
210   ! Make a label "SCOUNT," give the label a positive polarity, and
220   ! assign the lower 8 bits.
230   !
240   OUTPUT 707;":MACHINE1:SFORMAT:REMOVE ALL"
250   OUTPUT 707;":MACHINE1:SFORMAT:LABEL 'SCOUNT', POS, 0,0,255"
260   !
270   ! ****************** SETUP THE TRIGGER SPECIFICATION ********************
280   ! The trigger specification will use five sequence levels with the trigger
290   ! level on level four.  Resource terms A through E, and RANGE1 will be
300   ! used to store only desired counts from the 8-bit ripple counter.
```

```
310   !
320   ! Display the state trigger menu.
330   !
340   OUTPUT 707;":MENU 1,3"
350   !
360   ! Create a 5 level trigger specification with the trigger on the
370   ! fourth level.
380   !
390   OUTPUT 707;":MACHINE1:STRIGGER:SEQUENCE 5,4"
400   !
410   ! Define pattern terms A, B, C, D, and E to be 11, 22, 33, 44 and 59
420   ! decimal respectively.
430   !
440   OUTPUT 707;":MACHINE1:STRIGGER:TERM A,'SCOUNT','11'"
450   OUTPUT 707;":MACHINE1:STRIGGER:TERM B,'SCOUNT','22'"
460   OUTPUT 707;":MACHINE1:STRIGGER:TERM C,'SCOUNT','33'"
470   OUTPUT 707;":MACHINE1:STRIGGER:TERM D,'SCOUNT','44'"
480   OUTPUT 707;":MACHINE1:STRIGGER:TERM E,'SCOUNT','59'"
490   !
500   ! Define a Range having a lower limit of 50 and an upper limit of 58.
510   !
520   OUTPUT 707;":MACHINE1:STRIGGER:RANGE1 'SCOUNT','50','58'"
530   !
540   ! ***************** CONFIGURE SEQUENCE LEVEL 1 ***************************
550   ! Store NOSTATE in level 1 and Then find resource term "A" once.
560   !
570   OUTPUT 707;":MACHINE1:STRIGGER:STORE1 'NOSTATE'"
580   OUTPUT 707;":MACHINE1:STRIGGER:FIND1 'A',1"
590   !
600   ! ***************** CONFIGURE SEQUENCE LEVEL 2 ***************************
610   ! Store RANGE1 in level 2 and Then find resource term "E" once.
620   !
630   OUTPUT 707;":MACHINE1:STRIGGER:STORE2 'IN_RANGE1'"
640   OUTPUT 707;":MACHINE1:STRIGGER:FIND2 'E',1"
650   !
660   ! ***************** CONFIGURE SEQUENCE LEVEL 3 ***************************
670   ! Store NOSTATE in level 3 and Then find term "B" once.
680   !
690   OUTPUT 707;":MACHINE1:STRIGGER:STORE3 'NOSTATE'"
700   OUTPUT 707;":MACHINE1:STRIGGER:FIND3 'B',1"
710   !
720   ! ***************** CONFIGURE SEQUENCE LEVEL 4 ***************************
730   ! Store a combination of resource terms (C or D or RANGE1) in level 4 and
740   ! Then Trigger on resource term "E."
750   !
```

```
760   OUTPUT 707;":MACHINE1:STRIGGER:STORE4 '(C OR D OR IN_RANGE1)'"
770   !
780   ! ************************ NOTE ***********************
790   !      The FIND command selects the trigger in the
800   !      sequence level specified as the trigger level.
810   ! *****************************************************
820   !
830   OUTPUT 707;":MACHINE1:STRIGGER:FIND4 'E',1"
840   !
850   ! ***************** CONFIGURE SEQUENCE LEVEL 5 ***************************
860   ! Store anystate on level 5
870   !
880   OUTPUT 707;":MACHINE1:STRIGGER:STORE5 'ANYSTATE'"
890   !
900   ! ***************** START ACQUISITION ************************************
910   ! Place the logic analyzer in single acquisition mode, then determine when
920   ! the acquisition is complete.
930   !
940   OUTPUT 707;":RMODE SINGLE"
950   !OUTPUT 707;"*CLS"
960   OUTPUT 707;":START"
970   !
980   ! ****************** CHECK FOR MEASUREMENT COMPLETE **********************
990   ! Enable the MESR register and query the register for a measurement
1000  ! complete condition.
1010  !
1020  OUTPUT 707;":SYSTEM:HEADER OFF"
1030  OUTPUT 707;":SYSTEM:LONGFORM OFF"
1040  !
1050  Status=0
1060  OUTPUT 707;":MESE1 1"
1070  OUTPUT 707;":MESR1?"
1080  ENTER 707;Status
1090  !
1100  ! Print the MESR register status.
1110  !
1120  CLEAR SCREEN
1130  PRINT "Measurement complete status is ";Status
1140  PRINT "0 = not complete, 1 = complete"
1150  ! Repeat the MESR query until measurement is complete.
1160  WAIT 1
1170  IF Status=1 THEN GOTO 1190
1180  GOTO 1070
1190  PRINT TABXY(30,15);"Measurement is complete"
1200  !
```

```
1210  ! ********************** VIEW THE RESULTS ****************************
1220  ! Display the State Listing and select a line number in the listing that
1230  ! allows you to see the beginning of the listing on the logic analyer
1240  ! display.
1250  !
1260  OUTPUT 707;":MACHINE1:SLIST:COLUMN 1, 'SCOUNT', DECIMAL"
1270  OUTPUT 707;":MENU 1,7"
1280  OUTPUT 707;":MACHINE1:SLIST:LINE -16"
1290  !
1300  END
```

Making a State Compare measurement

This program example acquires a state listing, copies the listing to the compare listing, acquires another state listing, and compares both listings to find differences.

This program is written in such a way you can run it with the HP E2433-60004 Logic Analyzer Training Board. This example is the same as the "State Compare" example in chapter 3 of the *HP E2433-90910 Logic Analyzer Training Guide*.

```
10    ! ********** STATE COMPARE EXAMPLE ********************************
20    !            for the HP 1660-series Logic Analyzers
30    !
40    !
50    !************** SELECT THE LOGIC ANALYZER ************************
60    ! Select the module slot in which the logic analyzer is installed.
70    ! Always a 1 for  the HP 1660A-series logic analyzers.
80    !
90    OUTPUT 707;":SELECT 1"
100   !
110   !************** CONFIGURE THE STATE ANALYZER ***********************
120   ! Name Machine 1 "STATE," configure Machine 1 as a state analyzer, and
130   ! assign pod 1 to Machine 1.
140   !
150   OUTPUT 707;":MACHINE1:NAME 'STATE'"
160   OUTPUT 707;":MACHINE1:TYPE STATE"
170   OUTPUT 707;":MACHINE1:ASSIGN 1"
180   !
190   ! ****************************************************************
200   ! Remove all labels previously set up, make a label "SCOUNT," specify
210   ! positive logic, and assign the lower 8 bits of pod 1 to the label.
220   !
230   OUTPUT 707;":MACHINE1:SFORMAT:REMOVE ALL"
240   OUTPUT 707;":MACHINE1:SFORMAT:LABEL 'SCOUNT', POS, 0,0,255"
250   !
260   ! ****************************************************************
270   ! Make the "J" clock the Master clock and specify the falling edge.
280   !
290   OUTPUT 707;":MACHINE1:SFORMAT:MASTER J, FALLING"
300   !
310   ! ****************************************************************
320   ! Specify two sequence levels, the trigger sequence level, specify
```

```
330   ! FF hex for the "a" term which will be the trigger term, and store
340   ! no states until the trigger is found.
350   !
360   OUTPUT 707;":MACHINE1:STRIGGER:SEQUENCE 2,1"
370   OUTPUT 707;":MACHINE1:STRIGGER:TERM A,'SCOUNT','#HFF'"
380   OUTPUT 707;":MACHINE1:STRIGGER:STORE1 'NOSTATE'"
390   OUTPUT 707;":MENU 1,3"
400   !
410   ! ****************************************************************
420   ! Change the displayed menu to the state listing and start the state
430   ! analyzer in repetitive mode.
440   !
450   OUTPUT 707;":MENU 1,7"
460   OUTPUT 707;":RMODE REPETITIVE"
470   OUTPUT 707;":START"
480   !
490   ! ****************************************************************
500   ! The logic analyzer is now running in the repetitive mode
510   ! and will remain in repetitive until the STOP command is sent.
520   !
530   PRINT "The logic analyzer is now running in the repetitive mode"
540   PRINT "and will remain in repetitive until the STOP command is sent."
550   PRINT
560   PRINT "Press CONTINUE"
570   PAUSE
580   !
590   !*****************************************************************************
600   ! Stop the acquisition and copy the acquired data to the compare reference
610   ! listing.
620   !
630   OUTPUT 707;":STOP"
640   OUTPUT 707;":MENU 1,10"
650   OUTPUT 707;":MACHINE1:COMPARE:MENU REFERENCE"
660   OUTPUT 707;":MACHINE1:COMPARE:COPY"
670   !
680   ! The logic analyzer acquistion is now stopped, the Compare menu
690   ! is displayed, and the data is now in the compare reference
700   ! listing.
710   !
720   !*****************************************************************************
730   ! Display line 4090 of the compare listing and start the analyzer
740   ! in a repetitive mode.
750   !
760   OUTPUT 707;":MACHINE1:COMPARE:LINE 4090"
770   OUTPUT 707;":START"
```

```
780   !
790   ! Line 4090 of the listing is now displayed at center screen
800   ! in order to show the last four states acquired.  In this
810   ! example, the last four states are stable.  However, in some
820   ! cases, the end points of the listing may vary thus causing
830   ! a false failure in compare.  To eliminate this problem, a
840   ! partial compare can be specified to provide predicable end
850   ! points of the data.
860   !
870   PRINT "Press CONTINUE to send the STOP command."
880   PAUSE
890   OUTPUT 707;":STOP"
900   !
910   !*************************************************************************
920   ! The end points of the compare can be fixed to prevent false failures.
930   ! In addition, you can use partial compare to compare only sections
940   ! of the state listing you are interested in comparing.
950   !
960   OUTPUT 707;":MACHINE1:COMPARE:RANGE PARTIAL, 0, 508"
970   !
980   ! The compare range is now from line 0 to +508
990   !
1000  !*************************************************************************
1010  ! Change the Glitch jumper settings on the training board so that the
1020  ! data changes, reacquire the data and compare which states are different.
1030  PRINT "Change the glitch jumper settings on the training board so that the"
1040  PRINT "data changes, reacquire the data and compare which states are
different."
1050  !
1060  PRINT "Press CONTINUE when you have finished changing the jumper."
1070  !
1080  PAUSE
1090  !
1100  !*************************************************************************
1110  ! Start the logic analyzer to acquire new data and then stop it to compare
1120  ! the data.  When the acquistion is stopped, the Compare Listing Menu will
1130  ! be displayed.
1140  !
1150  OUTPUT 707;":START"
1160  OUTPUT 707;":STOP"
1170  OUTPUT 707;":MENU 1,10"
1180  !
1190  !*************************************************************************
1200  ! Dimension strings in which the compare find query (COMPARE:FIND?)
1210  ! enters the line numbers and error numbers.
```

```
1220  !
1230  DIM Line$[20]
1240  DIM Error$[4]
1250  DIM Comma$[1]
1260  !
1270  ! ************************************************************************
1280  ! Display the Difference listing.
1290  !
1300  OUTPUT 707;":MACHINE1:COMPARE:MENU DIFFERENCE"
1310  !
1320  ! ************************************************************************
1330  ! Loop to query all 508 possible errors.
1340  !
1350  FOR Error=1 TO 508
1360  !
1370  ! Read the compare differences
1380  !
1390  OUTPUT 707;":MACHINE1:COMPARE:FIND? "&VAL$(Error)
1400  !
1410  ! ************************************************************************
1420  ! Format the Error$ string data for display on the controller screen.
1430  !
1440  IF Error99 THEN GOTO 1580
1450  IF Error9 THEN GOTO 1550
1460  !
1470  ENTER 707 USING "#,1A";Error$
1480  ENTER 707 USING "#,1A";Comma$
1490  ENTER 707 USING "K";Line$
1500  Error_return=IVAL(Error$,10)
1510  IF Error_return=0 THEN GOTO 1820
1520  !
1530  GOTO 1610
1540  !
1550  ENTER 707 USING "#,3A";Error$
1560  ENTER 707 USING "K";Line$
1570  GOTO 1610
1580  !
1590  ENTER 707 USING "#,4A";Error$
1600  ENTER 707 USING "K";Line$
1610  !
1620  ! ************************************************************************
1630  ! Test for the last error.  The error number of the last error is the same
1640  ! as the error number of the first number after the last error.
1650  !
1660  Error_line=IVAL(Line$,10)
```

```
1670  IF Error_line=Error_line2 THEN GOTO 1780
1680  Error_line2=Error_line
1690  !
1700  ! ********************************************************************************
1710  ! Print the error numbers and the corresponding line numbers on the
1720  ! controller screen.
1730  !
1740  PRINT "Error number ",Error," is on line number ",Error_line
1750  !
1760  NEXT Error
1770  !
1780  PRINT
1790  PRINT
1800  PRINT "Last error found"
1810  GOTO 1850
1820  PRINT "No errors found"
1830  !
1840  !
1850  END
```

Transferring the logic analyzer configuration

This program uses the `SYSTem:SETup` query to transfer the configuration of the logic analyzer to your controller. This program also uses the `SYSTem:SETup` command to transfer a logic analyzer configuration from the controller back to the logic analyzer. The configuration data will set up the logic analyzer according to the data. It is useful for getting configurations for setting up the logic analyzer by the controller. This query differs from the `SYSTem:DATA` query because it only transfers the configuration and not the acquired data. The `SYSTem:SETup` command differs from the `SYSTem:DATA` command because it only transfers the configuration and not acquired data.

```
10    ! ****************** SETUP COMMAND AND QUERY EXAMPLE ********************
20    !                  for the HP 1660-series logic analyzers
30    !
***   ! ********************* CREATE TRANSFER BUFFER ************************
50    ! Create a buffer large enough for the block data.  See page 26-9 for
55    ! maximum block length.
56    !
60    ASSIGN @Buff TO BUFFER [170000]
70    !
80    ! ***************** INITIALIZE HPIB DEFAULT ADDRESS *********************
90    !
100   REAL Address
110   Address=707
120   ASSIGN @Comm TO Address
130   !
140   CLEAR SCREEN
150   !
160   ! ************* INTITIALIZE VARIABLE FOR NUMBER OF BYTES *****************
170   ! The variable "Numbytes" contains the number of bytes in the buffer.
180   !
190   REAL Numbytes
200   Numbytes=0
210   !
220   ! ************** RE-INITIALIZE TRANSFER BUFFER POINTERS ******************
230   !
240   CONTROL @Buff,3;1
250   CONTROL @Buff,4;0
260   !
270   ! ********************** SEND THE SETUP QUERY ***************************
```

```
280   OUTPUT 707;":SYSTEM:HEADER ON"
290   OUTPUT 707;":SYSTEM:LONGFORM ON"
300   OUTPUT @Comm;"SELECT 1"
310   OUTPUT @Comm;":SYSTEM:SETUP?"
320   !
330   ! ******************** ENTER THE BLOCK SETUP HEADER *********************
340   ! Enter the block setup header in the proper format.
350   !
360   ENTER @Comm USING "#,B";Byte
370   PRINT CHR$(Byte);
380   WHILE Byte<>35
390     ENTER @Comm USING "#,B";Byte
400     PRINT CHR$(Byte);
410   END WHILE
420   ENTER @Comm USING "#,B";Byte
430   PRINT CHR$(Byte);
440   Byte=Byte-48
450   IF Byte=1 THEN ENTER @Comm USING "#,D";Numbytes
460   IF Byte=2 THEN ENTER @Comm USING "#,DD";Numbytes
470   IF Byte=3 THEN ENTER @Comm USING "#,DDD";Numbytes
480   IF Byte=4 THEN ENTER @Comm USING "#,DDDD";Numbytes
490   IF Byte=5 THEN ENTER @Comm USING "#,DDDDD";Numbytes
500   IF Byte=6 THEN ENTER @Comm USING "#,DDDDDD";Numbytes
510   IF Byte=7 THEN ENTER @Comm USING "#,DDDDDDD";Numbytes
520   IF Byte=8 THEN ENTER @Comm USING "#,DDDDDDDD";Numbytes
530   PRINT Numbytes
540   !
550   ! ******************** TRANSER THE SETUP ********************************
560   ! Transfer the setup from the logic analyzer to the buffer.
570   !
580   TRANSFER @Comm TO @Buff;COUNT Numbytes,WAIT
600   !
610   ENTER @Comm USING "-K";Length$
620   PRINT "LENGTH of Length string is";LEN(Length$)
630   !
640   PRINT "**** GOT THE SETUP ****"
650   PAUSE
660   ! ********************* SEND THE SETUP **********************************
670   ! Make sure buffer is not empty.
680   !
690   IF Numbytes=0 THEN
700     PRINT "BUFFER IS EMPTY"
710     GOTO 1170
720   END IF
730   !
```

```
740   ! ********************* SEND THE SETUP COMMAND *************************
750   ! Send the Setup command
760   !
770   OUTPUT @Comm USING "#,15A";":SYSTEM:SETUP #"
780   PRINT "SYSTEM:SETUP command has been sent"
790   PAUSE
800   !
810   ! ********************* SEND THE BLOCK SETUP ***************************
820   ! Send the block setup header to the logic analyzer in the proper format.
830   !
840   Byte=LEN(VAL$(Numbytes))
850   OUTPUT @Comm USING "#,B";(Byte+48)
860   IF Byte=1 THEN OUTPUT @Comm USING "#,A";VAL$(Numbytes)
870   IF Byte=2 THEN OUTPUT @Comm USING "#,AA";VAL$(Numbytes)
880   IF Byte=3 THEN OUTPUT @Comm USING "#,AAA";VAL$(Numbytes)
890   IF Byte=4 THEN OUTPUT @Comm USING "#,AAAA";VAL$(Numbytes)
900   IF Byte=5 THEN OUTPUT @Comm USING "#,AAAAA";VAL$(Numbytes)
910   IF Byte=6 THEN OUTPUT @Comm USING "#,AAAAAA";VAL$(Numbytes)
920   IF Byte=7 THEN OUTPUT @Comm USING "#,AAAAAAA";VAL$(Numbytes)
930   IF Byte=8 THEN OUTPUT @Comm USING "#,AAAAAAAA";VAL$(Numbytes)
940   !
950   ! ************************ SAVE BUFFER POINTERS ************************
960   ! Save the transfer buffer pointer so it can be restored after the
970   ! transfer.
980   !
990   STATUS @Buff,5;Streg
1000  !
1010  ! ******************* TRANSFER SETUP TO THE HP 16550 ********************
1020  ! Transfer the setup from the buffer to the HP 1660A.
1030  !
1040  TRANSFER @Buff TO @Comm;COUNT Numbytes,WAIT
1050  !
1060  ! ********************** RESTORE BUFFER POINTERS ************************
1070  ! Restore the transfer buffer pointer
1080  !
1090  CONTROL @Buff,5;Streg
1100  !
1110  ! ********************* SEND TERMINATING LINE FEED *********************
1120  ! Send the terminating linefeed to properly terminate the setup string.
1130  !
1140  OUTPUT @Comm;""
1150  !
1160  PRINT "**** SENT THE SETUP ****"
1170  END
```

Transferring the logic analyzer acquired data

This program uses the `SYSTem:DATA` query to transfer acquired data to your controller. It is useful for getting acquired data for setting up the logic analyzer by the controller at a later time. This query differs from the `SYSTem:SETup` query because it transfers only the acquired data.

This program also uses the `SYSTem:DATA` command to transfer the logic analyzer data from the controller back to the logic analyzer and load the analyzer with the acquired data. The `SYSTem:DATA` command differs from the `SYSTem:SETup` command because it transfers both the configuration and the acquired data.

> You should always precede the SYSTem:DATA query and command with the SYSTem:SETup query and command if the acquired data depends on a specific configuration. If you are only interested in the acquired data for post processing in the controller and the data is not dependent on the configuration, you can use the SYSTem:DATA query and command alone.

```
10    ! ****************** DATA COMMAND AND QUERY EXAMPLE ********************
20    !                  for the HP 1660-series logic analyzers
30    !
40    ! ********************** CREATE TRANSFER BUFFER *************************
50    ! Create a buffer large enough for the block data.  See page 26-1 for
55    ! maximum block length.
56    !
60    ASSIGN @Buff TO BUFFER [170000]
70    !
80    ! ***************** INITIALIZE HPIB DEFAULT ADDRESS **********************
90    !
100   REAL Address
110   Address=707
120   ASSIGN @Comm TO Address
130   !
140   CLEAR SCREEN
150   !
160   ! ************* INTITIALIZE VARIABLE FOR NUMBER OF BYTES *****************
170   ! The variable "Numbytes" contains the number of bytes in the buffer.
180   !
190   REAL Numbytes
```

```
200   Numbytes=0
210   !
220   ! ************** RE-INITIALIZE TRANSFER BUFFER POINTERS ******************
230   !
240   CONTROL @Buff,3;1
250   CONTROL @Buff,4;0
260   !
270   ! ********************** SEND THE DATA QUERY **************************
280   OUTPUT 707;":SYSTEM:HEADER ON"
290   OUTPUT 707;":SYSTEM:LONGFORM ON"
300   OUTPUT @Comm;"SELECT 1"
310   OUTPUT @Comm;":SYSTEM:DATA?"
320   !
330   ! ******************** ENTER THE BLOCK DATA HEADER ********************
340   ! Enter the block data header in the proper format.
350   !
360   ENTER @Comm USING "#,B";Byte
370   PRINT CHR$(Byte);
380   WHILE Byte<>35
390     ENTER @Comm USING "#,B";Byte
400     PRINT CHR$(Byte);
410   END WHILE
420   ENTER @Comm USING "#,B";Byte
430   PRINT CHR$(Byte);
440   Byte=Byte-48
450   IF Byte=1 THEN ENTER @Comm USING "#,D";Numbytes
460   IF Byte=2 THEN ENTER @Comm USING "#,DD";Numbytes
470   IF Byte=3 THEN ENTER @Comm USING "#,DDD";Numbytes
480   IF Byte=4 THEN ENTER @Comm USING "#,DDDD";Numbytes
490   IF Byte=5 THEN ENTER @Comm USING "#,DDDDD";Numbytes
500   IF Byte=6 THEN ENTER @Comm USING "#,DDDDDD";Numbytes
510   IF Byte=7 THEN ENTER @Comm USING "#,DDDDDDD";Numbytes
520   IF Byte=8 THEN ENTER @Comm USING "#,DDDDDDDD";Numbytes
530   PRINT Numbytes
540   !
550   ! ******************** TRANSER THE DATA ********************************
560   ! Transfer the data from the logic analyzer to the buffer.
570   !
580   TRANSFER @Comm TO @Buff;COUNT Numbytes,WAIT
600   !
610   ENTER @Comm USING "-K";Length$
620   PRINT "LENGTH of Length string is";LEN(Length$)
630   !
640   PRINT "**** GOT THE DATA ****"
650   PAUSE
```

```
660   ! ********************* SEND THE DATA *********************************
670   ! Make sure buffer is not empty.
680   !
690   IF Numbytes=0 THEN
700     PRINT "BUFFER IS EMPTY"
710     GOTO 1170
720   END IF
730   !
740   ! ********************* SEND THE DATA COMMAND *************************
750   ! Send the Setup command
760   !
770   OUTPUT @Comm USING "#,14A";":SYSTEM:DATA #"
780   PRINT "SYSTEM:DATA command has been sent"
790   PAUSE
800   !
810   ! ********************* SEND THE BLOCK DATA ***************************
820   ! Send the block data header to the logic analyzer in the proper format.
830   !
840   Byte=LEN(VAL$(Numbytes))
850   OUTPUT @Comm USING "#,B";(Byte+48)
860   IF Byte=1 THEN OUTPUT @Comm USING "#,A";VAL$(Numbytes)
870   IF Byte=2 THEN OUTPUT @Comm USING "#,AA";VAL$(Numbytes)
880   IF Byte=3 THEN OUTPUT @Comm USING "#,AAA";VAL$(Numbytes)
890   IF Byte=4 THEN OUTPUT @Comm USING "#,AAAA";VAL$(Numbytes)
900   IF Byte=5 THEN OUTPUT @Comm USING "#,AAAAA";VAL$(Numbytes)
910   IF Byte=6 THEN OUTPUT @Comm USING "#,AAAAAA";VAL$(Numbytes)
920   IF Byte=7 THEN OUTPUT @Comm USING "#,AAAAAAA";VAL$(Numbytes)
930   IF Byte=8 THEN OUTPUT @Comm USING "#,AAAAAAAA";VAL$(Numbytes)
940   !
950   ! ********************** SAVE BUFFER POINTERS *************************
960   ! Save the transfer buffer pointer so it can be restored after the
970   ! transfer.
980   !
990   STATUS @Buff,5;Streg
1000  !
1010  ! ************** TRANSFER DATA TO THE LOGIC ANALYZER *****************
1020  ! Transfer the data from the buffer to the logic analyzer.
1030  !
1040  TRANSFER @Buff TO @Comm;COUNT Numbytes,WAIT
1050  !
1060  ! ********************** RESTORE BUFFER POINTERS **********************
1070  ! Restore the transfer buffer pointer
1080  !
1090  CONTROL @Buff,5;Streg
1100  !
```

```
1110  ! ******************** SEND TERMINATING LINE FEED **********************
1120  ! Send the terminating linefeed to properly terminate the data string.
1130  !
1140  OUTPUT @Comm;""
1150  !
1160  PRINT "**** SENT THE DATA ****"
1170  END
```

Checking for measurement completion

This program can be appended to or inserted into another program when you need to know when a measurement is complete. If it is at the end of a program it will tell you when measurement is complete. If you insert it into a program, it will halt the program until the current measurement is complete.

This program is also in the state analyzer example program in "Making a State Analyzer Measurement" on pages 27-7 and 27-8. It is included in the state analyzer example program to show how it can be used in a program to halt the program until measurement is complete.

```
420   ! ****************** CHECK FOR MEASUREMENT COMPLETE **********************
430   ! Enable the MESR register and query the register for a measurement
440   ! complete condition.
450   !
460   OUTPUT 707;":SYSTEM:HEADER OFF"
470   OUTPUT 707;":SYSTEM:LONGFORM OFF"
480   !
490   Status=0
500   OUTPUT 707;":MESE1 1"
510   OUTPUT 707;":MESR1?"
520   ENTER 707;Status
530   !
540   ! Print the MESR register status.
550   !
560   CLEAR SCREEN
570   PRINT "Measurement complete status is ";Status
580   PRINT "0 = not complete, 1 = complete"
590   ! Repeat the MESR query until measurement is complete.
600   WAIT 1
610   IF Status=1 THEN GOTO 630
620   GOTO 510
630   PRINT TABXY(30,15);"Measurement is complete"
640   !
650   END
```

Sending queries to the logic analyzer

This program example contains the steps required to send a query to the logic analyzer. Sending the query alone only puts the requested information in an output buffer of the logic analyzer. You must follow the query with an **ENTER** statement to transfer the query response to the controller. When the query response is sent to the logic analyzer, the query is properly terminated in the logic analyer. If you send the query but fail to send an **ENTER** statement, the logic analyzer will display the error message "Query Interrupted" when it receives the next command from the controller, and, the query response is lost.

```
10    !************************ QUERY EXAMPLE ************************
20    !                for the HP 1660-series Logic Analyzers
30    !                          .
40    ! ************************ OPTIONAL ****************************
50    ! The following two lines turn the headers and longform on so
60    ! that the query name, in its long form, is included in the
70    ! query response.
80    !
90    !          ************** NOTE ***************
100   !          If your query response includes real
110   !          or integer numbers that you may want
120   !          to do statistics or math on later, you
130   !          should turn both header and longform
140   !          off so only the number is returned.
150   !          ************************************
160   !
170   OUTPUT 707;":SYSTEM:HEADER ON"
180   OUTPUT 707;":SYSTEM:LONGFORM ON"
190   !
200   ! ************************************************************
210   ! Select the slot in which the logic analyzer is located.
220   ! Always a 1 for the HP 1660-series logic analyzers.
230   OUTPUT 707;":SELECT 1"
240   !
250   ! ************************************************************************
260   ! Dimension a string in which the query response will be entered.
270   !
280   DIM Query$[100]
290   !
300   ! ************************************************************
```

```
310   ! Send the query.  In this example the MENU? query is sent.  All
320   ! queries except the SYSTem:DATA and SYSTem:SETup can be sent with
330   ! this program.
340   !
350   OUTPUT 707;"MENU?"
360   !
370   ! ****************************************************************
380   ! The two lines that follow transfer the query response from the
390   ! query buffer to the controller and then print the response.
400   !
410   ENTER 707;Query$
420   PRINT Query$
430   !
440   !
450   END
```

Getting ASCII Data with PRINt? ALL Query

This program example shows you how to get ASCII data from a state listing using the `PRINt? ALL` query. There are two things you must keep in mind:

- You must select the logic analyzer, which is always `SELECT 1` for the HP 1660-series logic analyzers.
- You must select the proper menu. The only menus that allow you to use the `PRINt? ALL` query are the listing menus and the disk menu.

```
10    !                ****** ASCII DATA *******
20    !
30    !
40    ! This program gets STATE Listing data from the HP 1660-series logic
50    ! analyzers in ASCII form by using the PRINT? ALL query.
60    !
70    !*****************************************************************
80    !
90    DIM Block$[32000]
100   OUTPUT 707;"EOI ON"
110   OUTPUT 707;":SYSTEM:HEAD OFF"
120   OUTPUT 707;":SELECT 1"  ! Always a 1 for the HP 1660-series logic
130                           ! analyzers.
140   !
150   OUTPUT 707;":MENU 1,7"  ! Selects the Listing 1 menu.  Print? All
160                           ! will only work in Listing and Disk menus.
170   !
180   OUTPUT 707;":SYSTEM:PRINT? ALL"
190   ENTER 707 USING "-K";Block$
200   !
210   !*****************************************************************
220   ! Now display the ASCII data you received.
230   !
240   PRINT USING "K";Block$
250   !
260   END
```

Reading the disk with the CATalog? ALL query

The following example program reads the catalog of the disk currently in the logic analyzer disk drive. The `CATALOG? ALL` query returns the entire 70-character field. Because DOS directory entries are 70 characters long, you should use the `CATALOG? ALL` query with DOS disks.

```
10    !              ****** DISK CATALOG ******
20    !               using the CATALOG? query
30    !
40    DIM File$[100]
50    DIM Specifier$[2]
60    OUTPUT 707;":EOI ON"
70    OUTPUT 707;":SYSTEM:HEADER OFF"
80    !
90    OUTPUT 707;":MMEMORY:CATALOG? ALL" ! send CATALOG? ALL query
100   !
110   ENTER 707 USING "#,2A";Specifier$  ! read in #8
120   ENTER 707 USING "#,8D";Length      ! read in block length
130   !
140   ! Read and print each file in the directory
150   !
160   FOR I=1 TO Length STEP 51
170     ENTER 707 USING "#,51A";File$
180     PRINT File$
190   NEXT I
200   ENTER 707 USING "A";Specifier$     ! read in final line feed
210   END
```

Reading the Disk with the CATalog? Query

This example program uses the **CATALOG?** query without the **ALL** option to read the catalog of the disk currently in the logic analyzer disk drive. However, if you do not use the **ALL** option, the query only returns a 51-character field. Keep in mind if you use this program with a DOS disk, each filename entry will be truncated at 51 characters.

```
10    !               ****** DISK CATALOG ******
20    !                using the CATALOG? query
30    !
40    DIM File$[100]
50    DIM Specifier$[2]
60    OUTPUT 707;":EOI ON"
70    OUTPUT 707;":SYSTEM:HEADER OFF"
80    !
90    OUTPUT 707;":MMEMORY:CATALOG?"        ! send CATALOG? query
100   !
110   ENTER 707 USING "#,2A";Specifier$     ! read in #8
120   ENTER 707 USING "#,8D";Length         ! read in block length
130   !
140   ! Read and print each file in the directory
150   !
160   FOR I=1 TO Length STEP 51
170     ENTER 707 USING "#,51A";File$
180     PRINT File$
190   NEXT I
200   ENTER 707 USING "A";Specifier$        ! read in final line feed
210   END
```

Printing to the disk

This program prints acquired data to a disk file. The file can be either on a LIF or DOS disk. If you print the file to a DOS disk, you will be able to view the file on a DOS compatible computer using any number of file utility programs.

```
10    !        ********* PRINTING TO A DISK FILE **********
20    !
30    !
40    ! This program prints the acquired data to a disk file.  I will
50    ! print to either a LIF or DOS file using the PRINT ALL command.
60    !
70    !*****************************************************************
80    !
90    OUTPUT 707;":SELECT 1"  ! Always a 1 for the HP 1660-series logic
100                           ! analyzers.
110   !
120   OUTPUT 707;":MENU 1,7"  ! Selects the Listing 1 menu.  Print to disk
130                           ! will only work in Listing and Disk menus.
140   !
150   OUTPUT 707;":SYSTEM:PRINT ALL, DISK, 'DISKFILE'"
160   !
170   !*****************************************************************
180   ! Now display catalog to see that the file has been saved on the disk.
190   !
200   DIM File$[100]
210   DIM Specifier$[2]
220   OUTPUT 707;":EOI ON"
230   OUTPUT 707;":SYSTEM:HEADER OFF"
240   OUTPUT 707;":MMEMORY:CATALOG? ALL"
250   ENTER 707 USING "#,2A";Specifier$
260   ENTER 707 USING "#,8D";Length
270   FOR I=1 TO Length STEP 70
280     ENTER 707 USING "#,70A";File$
290     PRINT File$
300   NEXT I
310   ENTER 707 USING "A";Specifier$
320   END
```

Index

Warranty

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

Hewlett-Packard makes no warranty of any kind with regard to this material, including, but not limited to, the implied warranties of merchantability or fitness for a particular purpose.

Hewlett-Packard shall not be liable for errors contained herein or for damages in connection with the furnishing, performance, or use of this material.

This Hewlett-Packard product has a warranty against defects in material and workmanship for a period of one year from date of shipment. During the warranty period, Hewlett-Packard Company will, at its option, either repair or replace products that prove to be defective.

For warranty service or repair, this product must be returned to a service facility designated by Hewlett-Packard.

For products returned to Hewlett-Packard for warranty service, the Buyer shall prepay shipping charges to Hewlett-Packard and Hewlett-Packard shall pay shipping charges to return the product to the Buyer. However, the Buyer shall pay all shipping charges, duties, and taxes for products returned to Hewlett-Packard from another country.

Limitation of Warranty

The foregoing warranty shall not apply to defects resulting from improper or inadequate maintenance by the 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. Hewlett-Packard specifically disclaims the implied warranties of merchantability or fitness for a particular purpose.

Exclusive Remedies

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

Assistance

Product maintenance agreements and other customer assistance agreements are available for Hewlett-Packard products.

For any assistance, contact your nearest Hewlett-Packard Sales Office.

Safety

Do not install substitute parts or perform any unauthorized modification to the instrument.

Warning

The Warning symbol calls attention to a procedure, practice, or the like, which, if not correctly performed or adhered to, could result in personal injury. Do not proceed beyond a Warning symbol until the indicated conditions are fully understood and met.

Caution

The Caution symbol calls attention to an operating procedure, practice, or the like, which, if not correctly performed or adhered to, could result in damage to or destruction of part or all of the product. Do not proceed beyond a Caution symbol until the indicated conditions are fully understood or met.

Hewlett-Packard
P.O. Box 2197
1900 Garden of the Gods Road
Colorado Springs, CO 80901

About this edition

This is the first edition of the HP 01660-90902 Programmer's Guide for the HP 1660-Series Logic Analyzers. Edition dates are as follows:

1st edition, August 1992.

New editions are complete revisions of the manual. Update packages, which are issued between editions, contain additional and replacement pages to be merged into the manual by you. The dates on the title page change only when a new edition is published.

Many product updates and fixes do not require manual changes; and, conversely, manual corrections may be done without accompanying product changes. Therefore, do not expect a one-to-one correspondence between product updates and manual updates.

The following list of pages gives the date of the current edition and of any changed pages to that edition. Within the manual, any page changed since the last edition is indicated by printing the date the changes were made on the bottom of the page. If an update is incorporated when a new edition of the manual is printed, the change dates are removed from the bottom of the pages and the new edition date is listed on the title page.

August 1992: All pages original edition

Hewlett-Packard
Printed in the USA