

ARTISAN®
TECHNOLOGY GROUP

Your **definitive** source for quality pre-owned equipment.

Artisan Technology Group
(217) 352-9330 | sales@artisantg.com | artisantg.com

Full-service, independent repair center
with experienced engineers and technicians on staff.

We buy your excess, underutilized, and idle equipment
along with credit for buybacks and trade-ins.

Custom engineering
so your equipment works exactly as you specify.

- Critical and expedited services
- In stock / Ready-to-ship
- Leasing / Rentals / Demos
- ITAR-certified secure asset solutions

Expert team | Trust guarantee | 100% satisfaction

All trademarks, brand names, and brands appearing herein are the property of their respective owners.

Visit our website - *Click HERE*

Programmer's Guide

Publication number 16500-97018
First edition, December 1996

For Safety information, Warranties, and Regulatory information, see the pages behind the Index

© Copyright Hewlett-Packard Company 1987, 1990, 1993, 1994, 1996
All Rights Reserved

HP 16500C/16501A Logic Analysis System

In This Book

This programmer's guide contains general information, mainframe level commands, and programming examples for programming the HP 16500C/16501A Logic Analysis System. This guide focuses on how to program the system over the HP-IB interface, but also briefly explains how to use the RS-232-C and LAN interfaces. The Logic Analysis System cannot be programmed over the 16505 interface.

This guide provides a complete set of programming information for your system.

Organization

When you received your HP 16500C Programmer's Guide you received two binders, Volume 1 and Volume 2. The Volume 2 binder gives you a place to insert the module programmer's guides when the Volume 1 binder is full.

As you purchase additional measurement modules, insert their programmer's guides in the back of this binder or in the second binder.

What is in the HP 16500C/16500A Programmer's Guide?

The *HP 16500C/16501A Programmer's Guide* is organized in three parts.

1 Introduction to Programming

2 Programming Over HP-IB

3 Programming Over RS-232-C

4 Programming Over LAN

5 Programming and Documentation Conventions

6 Message Communication and System Functions

7 Status Reporting

8 Error Messages

9 Common Commands

10 Mainframe Commands

11 SYSTem Subsystem

12 MMEMory Subsystem

13 INTermodule Subsystem

14 TGTctrl Subsystem

15 Programming Examples

Part 1 Part 1 consists of chapters 1 through 8 and contains general information about programming basics, HP-IB, RS-232-C, and LAN interface requirements, documentation conventions, status reporting, and error messages. If you are already familiar with IEEE 488.2 programming and HP-IB or RS-232-C, 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 to send queries and how to retrieve query results from the instrument.

Read either chapter 2, "Programming Over HP-IB," chapter 3, "Programming Over RS-232-C," or chapter 4, "Programming over LAN" for information concerning the physical connection between the HP 16500C/16501A Logic Analysis System and your controller.

Chapter 5, "Programming and Documentation Conventions," gives an overview of all instructions and also explains the notation conventions used in the syntax definitions and examples.

Chapter 6, "Message Communication and System Functions," provides an overview of the operation of instruments that operate in compliance with the IEEE 488.2 standard.

Chapter 7 explains status reporting and how it can be used to monitor the flow of your programs and measurement process.

Chapter 8 contains error message descriptions.

Part 2 Part 2, chapters 9 through 14, explain each command in the command set for the mainframe. These chapters are organized in subsystems with each subsystem representing a menu.

The commands explained in this part give you access to common commands, mainframe commands, system level commands, disk commands, intermodule measurement, and target control commands. This part is designed to provide a concise description of each command.

Part 3 Part 3, chapter 15, contains program examples of actual tasks that show you how to get started in programming the HP 16500C/ 16501A Logic Analysis System at the mainframe level. The complexity of your programs and the tasks they accomplish are limited only by your imagination. 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 over HP-IB, RS-232-C, or LAN.

Contents

Part 1 General Information

1 Introduction to Programming

Introduction 1–2

Talking to the Logic Analysis System 1–3

Talking to Individual System Modules 1–4
Initialization 1–4
Instruction Syntax 1–6
Output Command 1–6
Device Address 1–7
Instructions 1–7
Instruction Terminator 1–8
Header Types 1–9
Duplicate Keywords 1–10
Query Usage 1–11
Program Header Options 1–12
Parameter Data Types 1–13
Selecting Multiple Subsystems 1–15

Receiving Information from the Logic Analysis System 1–16

Response Header Options 1–17
Response Data Formats 1–18
String Variables 1–19
Numeric Base 1–20
Numeric Variables 1–20
Definite-Length Block Response Data 1–21
Multiple Queries 1–22
System Status 1–23

2 Programming Over HP-IB

Interface Capabilities 2–3
Command and Data Concepts 2–3
Talk/Listen Addressing 2–3
HP-IB Bus Addressing 2–4
Local, Remote, and Local Lockout 2–5
Bus Commands 2–6

3 Programming Over RS-232-C

Interface Operation 3–3
RS-232-C Cables 3–3
Minimum Three-Wire Interface with Software Protocol 3–4
Extended Interface with Hardware Handshake 3–5
Cable Examples 3–6
Configuring the Logic Analysis System Interface 3–7
Interface Capabilities 3–8
RS-232-C Bus Addressing 3–9
Lockout Command 3–10

4 Programming Over LAN

Communicating with the HP 16500C 4–3
LAN Addressing 4–3
Password Protection and File Protection 4–4
Permission Levels: Control and Data 4–4
Controlling the HP 16500C 4–5
Echoing Commands 4–6
Copying Command Files 4–7
Writing to `\system\program` from a Program 4–8
Sending Commands to the HP 16500C Socket 4–11
Lockout Command 4–13

5 Programming and Documentation Conventions

Truncation Rule 5–3
Infinity Representation 5–4
Sequential and Overlapped Commands 5–4
Response Generation 5–4
Syntax Diagrams 5–4
Notation Conventions and Definitions 5–5
The Command Tree 5–6
Tree Traversal Rules 5–8
Command Set Organization 5–10
Subsystems 5–10
Program Examples 5–12

6 Message Communication and System Functions

Protocols 6–3
Syntax Diagrams 6–5
Syntax Overview 6–7

7 Status Reporting

Event Status Register 7–4
Service Request Enable Register 7–4
Bit Definitions 7–4
Key Features 7–6
Serial Poll 7–8
Parallel Poll 7–9
Polling HP-IB Devices 7–11
Configuring Parallel Poll Responses 7–11
Conducting a Parallel Poll 7–12
Disabling Parallel Poll Responses 7–13
HP-IB Commands 7–13

8 Error Messages

Device Dependent Errors 8–3
Command Errors 8–3
Execution Errors 8–4
Internal Errors 8–4
Query Errors 8–5

Part 2 Commands

9 Common Commands

*CLS (Clear Status) 9–5
*ESE (Event Status Enable) 9–6
*ESR (Event Status Register) 9–7
*IDN (Identification Number) 9–9
*IST (Individual Status) 9–9
*OPC (Operation Complete) 9–11
*OPT (Option Identification) 9–12
*PRE (Parallel Poll Enable Register Enable) 9–13
*RST (Reset) 9–14
*SRE (Service Request Enable) 9–15
*STB (Status Byte) 9–16
*TRG (Trigger) 9–17
*TST (Test) 9–18
*WAI (Wait) 9–19

10 Mainframe Commands

BEEPer 10–6
CAPability 10–7
CARDcage 10–8
CESE (Combined Event Status Enable) 10–10
CESR (Combined Event Status Register) 10–11
EOI (End Or Identify) 10–13
LER (LCL Event Register) 10–13
LOCKout 10–14
MENU 10–15

MESE<N> (Module Event Status Enable) 10–16
MESR<N> (Module Event Status Register) 10–18
RMODe 10–19
RTC (Real-time Clock) 10–20
SELect 10–21
SETColor 10–23
STARt 10–24
STOP 10–25
XWINdow 10–26

11 SYSTem Subsystem

DATA 11–5
DSP (Display) 11–6
ERRor 11–7
HEADer 11–8
LONGform 11–9
PRINt 11–10
SETup 11–12

12 MMEMory Subsystem

AUToload 12–7
CATalog 12–8
CD (Change Directory) 12–9
COPY 12–10
DOWNload 12–11
IDENtify 12–13
INITialize 12–14
LOAD[:CONFig] 12–15
LOAD :IASSembler 12–16
MKDir (Make Directory) 12–17
MSI (Mass Storage Is) 12–18
PACK 12–19
PURGe 12–20
PWD (Present Working Directory) 12–21

REName 12–22
STORe [:CONFig] 12–23
UPLoad 12–24
VOLume 12–25

13 INTermodule Subsystem

:INTermodule 13–5
DELete 13–6
HTIMe 13–7
INPort 13–8
INSert 13–9
OUTDrive 13–10
OUTPolar 13–10
OUTType 13–11
PORTEDGE 13–12
PORTLEV 13–13
SKEW<N> 13–14
TREE 13–15
TTIMe 13–17

14 TGTctrl Subsystem

:TGTctrl 14–5
ALL 14–6
AVAILable 14–7
BITS 14–8
CURSTate 14–9
DRIVe 14–9
LASTstate 14–10
NAMe 14–11
PULse 14–12
SIGNal 14–12
SIGSTatus 14–13
STATEs 14–14
STEP 14–15
TOGgle 14–15
TYPe 14–16

Part 3 Programming Examples

15 Programming Examples

Transferring the Mainframe Configuration 15–3
Checking for Intermodule Measurement Completion 15–6
Sending Queries to the Logic Analysis System 15–7
Getting ASCII Data with PRINt? ALL Query 15–9
Reading the disk with the CATalog? ALL query 15–10
Reading the Disk with the CATalog? Query 15–11
Printing to the disk 15–12

Index

Part 1

1 Introduction to Programming 1-1
2 Programming Over HP-IB 2-1
3 Programming Over RS-232-C 3-1
4 Programming Over LAN 4-1
5 Programming and Documentation Conventions 5-1
6 Message Communication and System Functions 6-1
7 Status Reporting 7-1
8 Error Messages 8-1

General Information

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 Analysis System," concentrates on initializing the bus, program syntax and the elements of instruction syntax. The second section, "Receiving Information from the Logic Analysis System," discusses how queries are sent and how to retrieve query results from the system.

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 16500C Logic Analysis System. There are three general categories of use. You can:

- Set up the system and start measurements
- Retrieve setup information and measurement results from the measurement modules
- Send measurement data to the measurement modules

The instructions listed in this manual give you access to the functions of the mainframe. This programming reference is designed to provide a concise description of each instruction for the mainframe. Individual module instruction descriptions are in the *Programmer's Guide* for each respective module.

In general, computers acting as controllers communicate with the instrument by sending and receiving messages over a remote interface, such as HP-IB, RS-232-C, or Ethernet LAN.

When programming the HP 16500C with the HP 16501A Expansion Frame connected, most of the remote operation of the expansion frame is transparent. The only time a programming command is affected by the presence of the expansion frame is when the number of slots is specified or returned from a query.

Instructions for programming the system 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 system. For example, HP 9000 Series 300 BASIC uses the OUTPUT statement for sending commands and queries to the system. 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.

Talking to Individual System Modules

Talking to individual system modules within the HP 16500C Logic Analysis System is done by preceding the module commands with the SELECT command and the number of the slot in which the desired module is installed. The mainframe is selected in the same way as an installed module by using the SELECT 0 command.

Example

To select the module in slot 3 use the following:

```
OUTPUT XXX;":SELECT 3"
```

See Also

Chapter 10, "Mainframe Commands" for more information on the SELECT command.

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 Analysis System. The parser is the program resident in the Logic Analysis System that reads the instructions you send to it from the controller.

After clearing the interface, you could, for example, preset the logic analyzer module 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 system.

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

Example This program demonstrates a simple HP BASIC command structure used to program the Logic Analysis System.

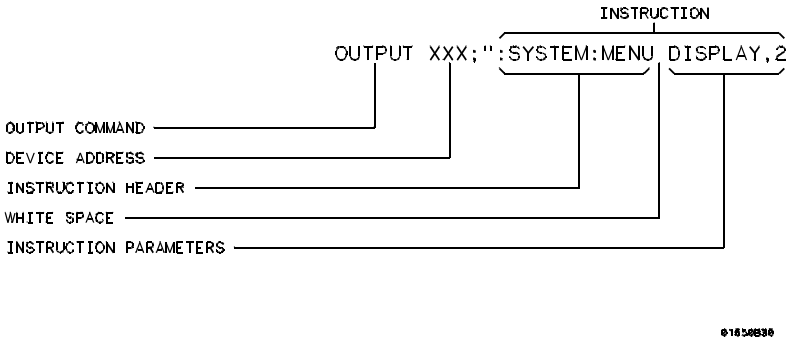
```
10 CLEAR XXX                          !Initialize instrument interface
20 OUTPUT XXX;":SYSTEM:HEADER ON"     !Turn headers on
30 OUTPUT XXX;":SYSTEM:LONGFORM ON"   !Turn long form on
40 DIM Card$[100]                     !Reserve memory for string variable
50 OUTPUT XXX;":CARDCAGE?"            !Verify which modules are loaded
60 ENTER XXX;Card$                    !Enter result in a string variable
70 PRINT Card$                        !Print result of query
80 OUTPUT XXX;":MMEM:LOAD:CONFIG 'TEST._E',5"  !Load configuration file
                                      !into module in slot E
90 OUTPUT XXX;":SELECT 5"             !Select module in slot E
100 OUTPUT XXX;":MENU 5,3:            !Select menu for module in slot E
110 OUTPUT XXX;":RMODE SINGLE"        !Select run mode
120 OUTPUT XXX;":START"               !Run the measurement
```

See Also Chapter 12, "MMEMory Subsystem" for more information on the LOAD command.

Instruction Syntax

To program the system 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 300 BASIC 6.2 is used in the programming examples, except where noted. 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 protocol you use, in addition to the actual address. If you are using HP-IB, see chapter 2, "Programming Over HP-IB." If you are using RS-232-C, see chapter 3, "Programming Over RS-232-C." If you are using Ethernet LAN, see chapter 4, "Programming Over LAN."

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 for the mainframe in figure 5-1 illustrates how all the keywords can be joined together to form a complete header (see chapter 5, "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 system.

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. 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.

Example

```
*CLS
```

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 analysis system instructions that are immediately followed by a question mark (?) are queries. After receiving a query, the Logic Analysis System parser places the response in the output buffer. The output message remains in the buffer until it is read or until another 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 system is currently configured. They are also used to get results of measurements made by the modules in the system.

Example

This instruction places the current full-screen time for machine 1 of the logic analyzer module in slot 2 in the output buffer.

```
:SELECT 2: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 **`:SELECT 2: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 from the Logic Analysis System."

Program Header Options

Program headers can be sent using any combination of uppercase or lowercase ASCII characters. System 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 5, "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. The types 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 11); the **CATalog**, **UPLoad**, and **DOWNload** instructions in the **MMEMory** subsystem (see chapter 12). These syntax rules also show how data may be formatted when sent back from the system 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. Do not combine an exponent with a unit.

See Also Tables 6-1 and 6-2 in chapter 6, "Message Communications and System Functions," list all available suffixes.

Example The following numbers are all equal:

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

The system will recognize binary, octal, and hexadecimal base numbers. The base of a number is specified with a prefix. The recognized prefixes are #B for binary, #Q for octal, and #H for hexadecimal. The absence of a prefix indicates the number is decimal which is the default base.

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 can not be used interchangeably. 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 long form or short form (if one exists) may be used. Uppercase and lowercase letters may be mixed freely. When receiving responses, uppercase letters will be used exclusively. The use of long form or short form in a response depends on the setting you last specified via the :SYSTem:LONGform command.

Selecting Multiple Subsystems

You can send multiple program commands and program queries for different subsystems within the same selected module 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

```
:SELECT 2;:MACHINE1:ASSIGN2;:SYSTEM:HEADERS ON
```

Receiving Information from the Logic Analysis System

After receiving a query (logic analysis system instruction followed by a question mark), the system 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 the system'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 **:SYSTEM:LONGFORM?**, you must follow that query with an input statement. In BASIC, this is usually done with an ENTER statement and in C with a read command.

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** 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.

Example

The following examples show some possible responses for a **:SELECT 2:MACHINE1:SFORMAT:THRESHOLD2?** query:

with HEADER OFF:

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

with HEADER ON and LONGFORM OFF:

```
:SEL 2:MACH1:SFOR:THR2<white_space><data><terminator>
```

with HEADER ON and LONGFORM ON:

```
:SELECT 2:MACHINE1:SFORMAT:THRESHOLD2<white_space>
<data><terminator>
```

See Also

Chapter 11, "SYSTem Subsystem" for information on turning the HEADER and LONGFORM commands on and off.

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.

Example

The following are possible responses to the :**SELECT 2:MACHINE1: TFORMAT: LAB? 'ADDR'** query.

Header on; Longform on

```
:SELECT 2:MACHINE1:TFORMAT:LABEL "ADDR  ",19,
POSITIVE<terminator>
```

Header on;Longform off

```
:SEL 2:MACH1:TFOR:LAB "ADDR  ",19,POS<terminator>
```

Header off; Longform on

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

Header off; Longform off

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

See Also

The individual commands in Part 2 of this guide contain information on the format (string or numeric) of the data returned from each query.

String Variables

Because there are so many ways to code numbers, the HP 16500C Logic Analysis System 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 system, 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 slot 3. In BASIC, the **&** operator is used for string concatenation.

```
10 LET Machine$ = ":SELECT 3:MACHINE2"  !Send all instructions to machine 2 in
       !slot 3
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$
```

The output of the system 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 module data being returned to a string variable with headers off:

```
10 OUTPUT XXX;":SYSTEM:HEADER OFF"
20 DIM Rang$[30]
30 OUTPUT XXX;":SELECT 2:MACHINE1:TWAVEFORM:RANGE?"
40 ENTER XXX;Rang$
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 on screen. 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 accidentally trying to convert the header into a number.

Example

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

```
10 OUTPUT XXX;":SYSTEM:HEADER OFF"
20 OUTPUT XXX;":SELECT 2:MACHINE1:TWAVEFORM:RANGE?"
30 ENTER XXX;Rang
40 PRINT Rang
50 END
```

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

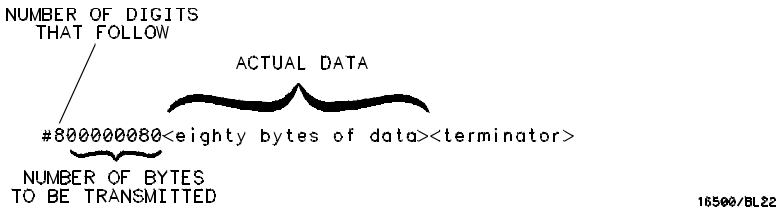
Definite-Length Block Response Data

Definite-length block response data, also referred 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 block 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 HP16500C Logic Analysis System.

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 system 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 BASIC 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 in HP BASIC is used to read the query :SYSTEM:HEADERS?;LONGFORM? into multiple numeric variables:

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

System Status

Status registers track the current status of the mainframe and the installed modules. By checking the system status, you can find out whether an operation has been completed, whether a module is receiving triggers, and more.

See Also

Chapter 7, "Status Reporting," explains how to check the status of the system and the installed modules.

2

Programming Over HP-IB

Introduction

This section describes the interface functions and some general concepts of HP-IB. In general, these functions are defined by IEEE 488.1 (HP-IB 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 16500C, 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 commands and responses found in chapters 10 through 14 of this guide for the mainframe and the respective *Programmer's Guides* for each module installed in the mainframe.

Talk/Listen Addressing

By using the touchscreen fields in the System Configuration menu, the HP-IB interface can be placed in talk-only mode by connecting to the printer or in addressed talk/listen mode by connecting to the controller.

See Also

Chapter 3, "Configuring Communications" in the *HP 16500C User's Reference*

Talk-only mode must be used when you want the system to talk directly to a printer without the aid of a controller. Addressed talk/listen mode is used when the system will operate in conjunction with a controller. When the system 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.

HP-IB Bus Addressing

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 analysis system will accept and execute bus commands while in local mode, and the front panel will also be entirely active. If the HP 16500C is in remote mode, the system will go from remote to local with any touchscreen, mouse, or keyboard 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 system to the power-on defaults and purges any acquired data in the acquisition memory of all the installed modules.

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. 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.

See Also

:SYSTem:LOCKout in chapter 10, "Mainframe Commands"

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 system.

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-232-C

Introduction

This chapter describes the interface functions and some general concepts of RS-232-C. The RS-232-C interface on this instrument is Hewlett-Packard's implementation of EIA Recommended Standard RS-232-C, *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 16500C Logic Analysis System can be programmed by a controller over RS-232-C 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 controlling an HP 16500C Logic Analysis System over RS-232-C, 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 interface, XON/XOFF must be used to handle protocol between the devices. For an extended hardwire interface, protocol may be handled either with XON/XOFF or by manipulating the CTS and RTS lines of the RS-232-C link. In all cases, the DCD and DSR lines to the Logic Analysis System must remain high for proper operation.

With extended hardwire operation, a high on the CTS line allows the Logic Analysis System to send data, and a low prevents the Logic Analysis System from transmitting data. 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 a three-wire interface has no control over the CTS line, internal pull-up resistors in the Logic Analysis System assure that this line remains high for proper three-wire operation.

RS-232-C Cables

The correct cable for the RS-232-C interface depends on your specific application and whether you use software or hardware handshake protocol. The following paragraphs describe which lines of the HP 16500C Logic Analysis System are used to control the handshake operation of RS-232-C relative to the system. To locate the proper cable for your application, refer to the reference manual for your computer or controller. It should describe the exact handshake protocol your controller can use to operate over an RS-232-C bus. In this chapter you will also find HP cable recommendations for hardware handshake.

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 Analysis System and the controller. Because the three-wire interface provides no hardware means to control data flow between the controller and the Logic Analysis System, only XON/OFF can control this data flow. The three-wire interface provides a much simpler connection between devices since you can ignore hardware handshake requirements.

> The communications software you are using in your computer/controller must be capable of using XON/XOFF exclusively in order to use three-wire interface cables. For example, some communications software packages can use XON/XOFF but also depend on the CTS and DSR lines being true to communicate.

The Logic Analysis System uses the following connections on its RS-232-C interface for three-wire communication:

- Pin 5 SGND (Signal Ground)
- Pin 3 TD (Transmit Data from Logic Analysis System)
- Pin 2 RD (Receive Data into Logic Analysis System)

The TD (Transmit Data) line from the Logic Analysis System must connect to the RD (Receive Data) line on the controller. Likewise, the RD line from the Logic Analysis System must connect to the TD line on the controller. Internal pull-up resistors in the Logic Analysis System 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 Analysis System and the controller. The Logic Analysis System uses the following connections on its RS-232-C interface for extended interface communication:

- Pin 5 SGND (Signal Ground)
- Pin 3 TD (Transmit Data from Logic Analysis System)
- Pin 2 RD (Receive Data into Logic Analysis System)

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

- Pin 7 RTS (Request To Send) is an output from the Logic Analysis System which can be used to control incoming data flow.
- Pin 8 CTS (Clear To Send) is an input to the Logic Analysis System which controls data flow from the Logic Analysis System.
- Pin 6 DSR (Data Set Ready) is an input to the Logic Analysis System which controls data flow from the Logic Analysis System within two bytes.
- Pin 1 DCD (Data Carrier Detect) is an input to the Logic Analysis System which controls data flow from the Logic Analysis System within two bytes.
- Pin 4 DTR (Data Terminal Ready) is an output from the Logic Analysis System which is enabled as long as the Logic Analysis System is turned on.

The TD (Transmit Data) line from the Logic Analysis System must connect to the RD (Receive Data) line on the controller. Likewise, the RD line from the Logic Analysis System must connect to the TD line on the controller.

The RTS (Request To Send) is an output from the Logic Analysis System 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 Analysis System, which control data flow from the Logic Analysis System. Internal pull-up resistors in the Logic Analysis System 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 along with the CTS line high to enable the Logic Analysis System to send data to the controller. A low on any one of these

lines will disable the Logic Analysis System data transmission. Pulling the CTS line low during data transmission will stop Logic Analysis System data transmission immediately. Pulling either the DSR or DCD line low during data transmission will stop Logic Analysis System data transmission, but as many as two additional bytes may be transmitted from the Logic Analysis System.

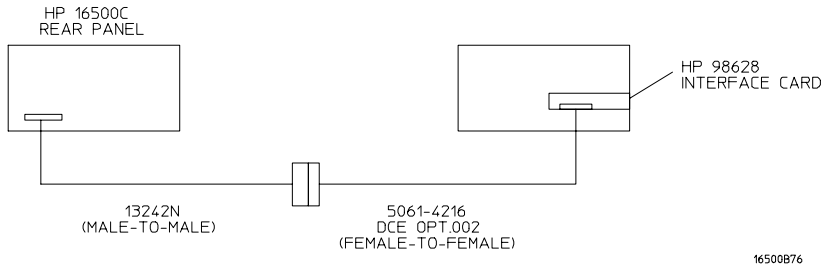
Cable Examples

HP 9000 Series 300

Figure 3-1 is an example of how to connect the HP 16500C Logic Analysis System to the HP 98628A Interface card of an HP 9000 series 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 Logic Analysis System.

Figure 3-1



Cable Example

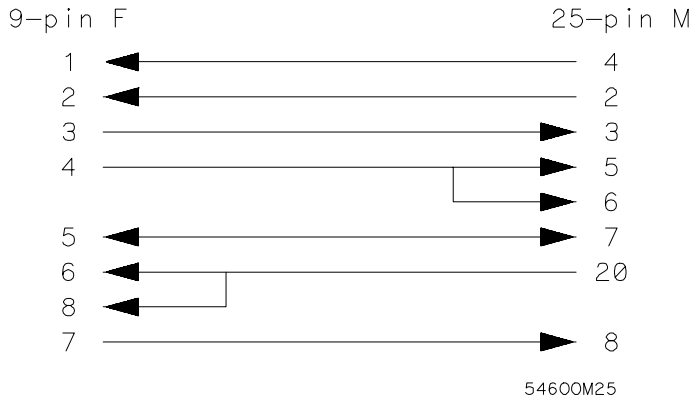
HP Vectra Personal Computers and Compatibles

Figure 3-2 gives an example of a cable that will work for the extended interface with hardware handshake. Keep in mind that this cable should work if your computer's serial interface supports the four common RS-232-C handshake signals as defined by the RS-232-C standard. The four common handshake signals are Data Carrier Detect (DCD), Data Terminal Ready (DTR), Clear to Send (CTS), and Ready to Send (RTS).

Figure 3-2 shows the schematic of a 9-pin female to 25-pin male cable. The following HP cables support this configuration:

- HP 24542G, DB-9(F) to DB-25(M), 3 meter
- HP 24542H, DB-9(F) to DB-25(M), 3 meter, shielded
- HP 45911-60009, DB-9(F) to DB-25(M), 1.5 meter

Figure 3-2



9-pin (F) to 25-pin (M) Cable

Configuring the Logic Analysis System Interface

The RS-232 Settings field in the System Configuration Menu allows you access to the RS-232 Settings menu where the RS-232-C interface is configured. If you are not familiar with how to configure the RS-232-C interface, refer to chapter 3, "Configuring Communications," in the *HP 16500C Logic Analysis System 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 Analysis System to properly communicate over the RS-232-C bus. The RS-232-C interface capabilities of the HP 16500C Logic Analysis System 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 Analysis System) controls data flow and can request that the sender (Logic Analysis System 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 16500C Logic Analysis System 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 16500C Logic Analysis System must be in the same bit mode to properly communicate over the RS-232-C. This means that the controller must have the capability to send and receive 8-bit data.

See Also

For more information on the RS-232-C interface, refer to the *HP 16500C Logic Analysis System User's Reference*. For information on RS-232-C voltage levels and connector pinouts, refer to the *HP 16500C Logic Analysis System Service Guide*.

RS-232-C Bus Addressing

The RS-232-C address you must use is dependent on the computer or controller you are using to communicate with the Logic Analysis System.

HP Vectra Personal Computers or compatibles

If you are using an HP Vectra Personal Computer or compatible, it must have an unused serial port to which you connect the Logic Analysis System's RS-232-C port. The proper address for the serial port is dependent on the hardware configuration of your computer. Additionally, your communications software must be configured to address the proper serial port. Refer to your computer and communications software manuals for more information on setting up your serial port address.

HP 9000 Series 300 Controllers

Each RS-232-C interface card for the HP 9000 Series 300 Controller 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-232-C interface select code is 9, the device address required to communicate over the RS-232-C 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 **`:LOCKout OFF`** command.

Hint

Cycling the power will also restore local control, but this will also reset certain RS-232-C states. It also resets the Logic Analysis System to the power-on defaults and purges any acquired data in the acquisition memory of all the installed modules.

See Also

For more information on this command see chapter 11, "SYSTem Subsystem."

4

Programming Over LAN

Introduction

This chapter describes different ways you can program your logic analysis system over a LAN. There are no commands needed for controlling the connection, and no special cabling issues. This chapter assumes you have already set up your LAN, and concentrates on how to control the HP 16500C from a host computer.

Communicating with the HP 16500C

You can communicate with the HP 16500C in several ways. If you NFS mount your logic analysis system, it behaves like a disk drive on your LAN and programs control it by writing to the `\system\program` file. The other common way to control the instrument is through telnet or another socket-style connection.

> The HP 16500C must be turned on and completely booted up before you can mount the system to your network. Once power is applied to the system and the System Configuration menu is displayed, allow an additional 15 seconds before attempting to connect to the system.

The LAN connection does not provide real-time programming control. Due to the message handling protocol of Ethernet LAN, messages take an indeterminate amount of time to reach their destinations. There can be no guarantee that commands sent from your computer will reach the HP 16500C in a timely way, although the majority of messages do.

LAN Addressing

All devices on an Ethernet LAN are uniquely identified by their Ethernet address. The Ethernet address is set in the hardware of any Ethernet-capable device.

However, the utilities you use to communicate with the HP 16500 Logic Analysis System follow the TCP/IP protocol. This protocol assigns unique IP addresses in software. When you connected your logic analysis system to your LAN, you or your system administrator assigned an IP address to the HP 16500C. Use this address to communicate with the HP 16500C. You can check the address by selecting LAN Settings in the System Configuration menu.

Password Protection and File Protection

There is no protection or security built into the HP 16500C. If you attempt to connect to the logic analysis system via FTP, and you are prompted for a password, leave the password field blank.

The operating system files, which are stored in the `\system` directory, are also not protected against accidental deletion. If these files are deleted, the HP 16500C will not operate the next time it is rebooted. If you do delete any of these files, copy them from the flexible disks labeled "16500 Operating System" back onto the hard disk, into the `\system` directory.

Permission Levels: Control and Data

The HP 16500C system can be mounted on your network with two different levels of access, "control" or "data." When you mount the HP 16500 system, you specify the type of access. The general syntax for mounting is:

UNIX
```
mount [host or IP address]:/[control|data] /[drive name]
```

DOS
```
net use [drive name] [host or IP address]:/[control|data]
net use [drive name] \\[host or IP address]\[control|data]
```

There are two differences between control and data permissions. First, the control level provides read and write access to all files. The data level provides write access only for the disk drives, and read access for all other files. Second, control allows you to send programming commands to the HP 16500C system, and data level does not.

> You must be connected as the control user to program the HP 16500C.

The HP 16500C will accommodate one data and one control user at a time. There can be only one control user at any time through any of the connection methods – NFS mount, ftp, telnet, or using a socket. For example, if you ftp to the HP 16500C as control, no one else can program it through any of the other methods.

Controlling the HP 16500C

To control the HP 16500C Logic Analysis System with programming commands, you can either write the commands to `\system\program`, or open a socket in a C program. Either way, the controller in the System Configuration menu must be set to LAN.

> In order to send programming commands to the HP 16500C \system\program file, the system must be connected to the LAN and you must be connected to the system as the control user.

The `\system\program` file

Once the logic analysis system is connected to your network, you can send commands to the system by sending them as text strings to the file location `\system\program`. You can send the strings using a variety of methods:

- echo a string from the command line to `\system\program`
- copy an ASCII file containing a series of commands to `\system\program`
- from within a C or BASIC program, open the file `\system\program` and write the commands to it using "fwrite" or "output".

Sockets

If you are programming in C or another language that supports sockets, you can write strings directly to the HP 16500C system. Socket connections are automatically control users, so if someone else is already connected to the logic analysis system as control user you will not be able to connect. You can also directly connect to the parser socket using telnet, and send commands interactively. All socket connections, including telnet, need to specify port or address 5025.

Echoing Commands

To send a command directly from the command line or prompt of your PC or workstation to the HP 16500C system, echo a text string containing the command to the file location `\system\program`.

> In order to send commands to the HP 16500C system parser, you must be connected to the system as the control user.

Example

To run the logic analyzer and acquire data, at the DOS prompt enter:

```
c:>echo :START > L:\system\program
```

If you are using a UNIX system, you can use the UNIX `echo` command.

Example

An HP 16550A state/timing analyzer is installed in slot C (slot 3) of your HP 16500C mainframe. To clear the trigger set-up on the HP 16550A, at the DOS prompt enter:

```
c:>echo :SELECT 3 > L:\system\program
c:>echo :MACHINE1:STRIGGER:CLEAR ALL > L:\system\program
```

If you are using a UNIX system, you can use the UNIX `echo` command. The first command selects the state/timing analyzer in slot C. The second command clears the trigger.

Copying Command Files

To control the HP 16500C system with longer sets of commands, you can first type the commands into an ASCII file. You then copy the file to the HP 16500 program file, at location `\system\program`. Files copied to this file location are passed on to the HP 16500C system's command parser.

Example

An HP 16550A state/timing analyzer is installed in slot C (slot 3) of your HP 16500C mainframe. To clear the format and trigger set-ups on the HP 16550A, using a program file, first type the commands into an ASCII text file.

File clear.txt:

```
:SELECT 3
:MACHINE1:SFORMAT:REMOVE ALL
:MACHINE1:STRIGGER:CLEAR ALL
```

The first command selects the state/timing analyzer in slot C to receive programming commands. The second command clears the format set-up. The third command clears the trigger set-up.

Now copy the file to the HP 16500 system. At the DOS prompt enter:

```
copy clear.txt L:\system\program
```

If you are using a UNIX system, you might use the `cp` command. In an MS-Windows environment, you can use File Manager.

Writing to `\system\program` from a Program

You can send commands to the HP 16500C program file from a program running on your PC or workstation. The basic procedure is to open the program file and send text strings containing the commands to the file. In C, you can use the `fwrite` or `putstr` commands to write text strings to the program file.

Your operating system may buffer the commands before sending them to the HP 16500C system. To prevent this, you may need to empty the buffer after each command. In C, you can use the `flush` command to empty the buffer.

Queries

Responses to queries appear as text strings in `\system\program`. To retrieve information from queries, create a text buffer, open the `program` file and read the contents of the file into the buffer. In C you can use the `fread` or `getstr` commands to read the contents of the file into the buffer. Whenever you send queries to the HP 16500 system, you will need to pause your program for a short time, to allow the system to process the query before you attempt to read the response. A time equal to or slightly greater than the file timeout is sufficient.

Resetting the File Pointer

Whenever you change from reading `\system\program` to writing to it, or from writing to reading, you will need to reset the file pointer to the beginning of the file. In C, you can use the `rewind` command to reset the pointer, or you can close the program file, then immediately re-open it.

Example

The following example in C opens the `\system\program` file and sends several commands and queries. Responses to queries appear as text strings in `\system\program`. The HP 16500C has been NFS-mounted in the `\users` directory.

```
#include <stdio.h>
#include <unistd.h>

#define STR_LEN 80

void putstr(FILE *file, char *str)
{
  fwrite(str, strlen(str), 1, file);
}

int getstr(FILE *file, char* str)
{
  return(fread(str, 1, STR_LEN, file));
}

void main()
{
  FILE *file;
  int num;
  char receive_str[STR_LEN];

/* Send a query and retrieve and print the response*/
  file = fopen("/users/system/program", "r");
  while (getstr(file, receive_str) != 0);
  fclose(file);
  file = fopen("/users/system/program", "w");
  putstr(file, "*idn?\n");
  fclose(file);
  sleep(1);
  file = fopen("/users/system/program", "r");
  while (getstr(file, receive_str) == 0);
  fclose(file);
  printf("%s\n", receive_str);
```

```
/*Send command strings to the HP16500*/
  file = fopen("/users/system/program", "w");
  putstr(file, "*rst\n");
  putstr(file, ":sel 4\n");
  putstr(file, ":mach1:twav:range 1 s\n");
  putstr(file, ":start\n");
  putstr(file, ":mach1:twav:range 100 ns\n");
  fclose(file);
  sleep(2);
  file = fopen("/users/system/program", "r");
  while (getstr(file, receive_str) == 0);
  fclose(file);
  printf("%s\n", receive_str);
}
```

Sending Commands to the HP 16500C Socket

If you are programming in C, you can use a socket to communicate with the HP 16500 system. By opening a socket connection, you can send program commands directly to the command parser. The HP 16500C system socket port identification number is 5025.

You can also connect directly to the parser socket and type commands directly to the HP 16500. The second example uses telnet to connect to the parser socket.

Example

The following C program opens a socket and sends a query to request the instrument's identity. If someone else is already connected as control user, the socket will eventually close without receiving a response.

```
#include <stdio.h>
#include <sys/types.h>
#include <sys/socket.h>
#include <netinet/in.h>

typedef struct sockaddr_in tdSOCKET_ADDR;

#define PARSER_PORT         5025
#define SERV_HOST_ADDR      "15.10.96.12"
#define PARSER_BUFFER_SIZE  100

char  receiveBuffer[PARSER_BUFFER_SIZE],
     *cmdString = { "*IDN?\r\n" };

main ()
{
              int  sockfd,
                   port;
    tdSOCKET_ADDR  serv_addr;
             char *addr;

/* Initialize a server socket */
    port = PARSER_PORT;
    addr = SERV_HOST_ADDR ;
    serv_addr.sin_family = AF_INET;
    serv_addr.sin_addr.s_addr = inet_addr ( addr );
    serv_addr.sin_port = htons ( port );
```

```
 /* Create an endpoint for communication */
     sockfd = socket( AF_INET, SOCK_STREAM, 0 );

 /* Initiate a connection on the created socket */
     connect( sockfd, ( tdSOCKET_ADDR * )&serv_addr,
              sizeof ( serv_addr ) );

 /* Send a message from the created socket */
     send ( sockfd, cmdString, strlen ( cmdString ), 0 );

/* Receive a message from the 16500 socket */
    recv ( sockfd, receiveBuffer, sizeof( receiveBuffer ),0 );
    printf ( "%s\n", receiveBuffer );
    close ( sockfd );
}
```

Example

This example uses telnet to connect directly to the HP 16500 parser socket. To remotely interact with the logic analyzer, enter:

```
telnet [symbolic name or IP address] 5025
```

You must specify the HP 16500 parser socket address 5025. You can now type commands directly to the HP 16500 system. The results of queries will appear on the command line of your PC or workstation.

To send the command which will run the analyzer and acquire data, enter:

```
:START
```

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 **`:LOCKout OFF`** command.

Hint

Cycling the power will also restore local control, but this will reset the Logic Analysis System to the power-on defaults and purges any acquired data in the acquisition memory of all the installed modules.

See Also

For more information on this command see chapter 11, "SYSTem Subsystem."

5

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 long form has four or fewer characters, there is no change in the short form. When the long form has more than four characters the short form 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 5-1.

Table 5-1 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 16500C Logic Analysis System 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 16500C Logic Analysis System 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 5-1) shows all commands in the HP 16500C Logic Analysis System and the relationship of the commands to each other. You should notice that the common commands are not actually connected to the other commands in the command tree. After a <NL> (linefeed - ASCII decimal 10) has been sent to the instrument, the parser will be set to the root of the command tree. Parameters are not shown in this figure. The command tree allows you to see what the system's 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

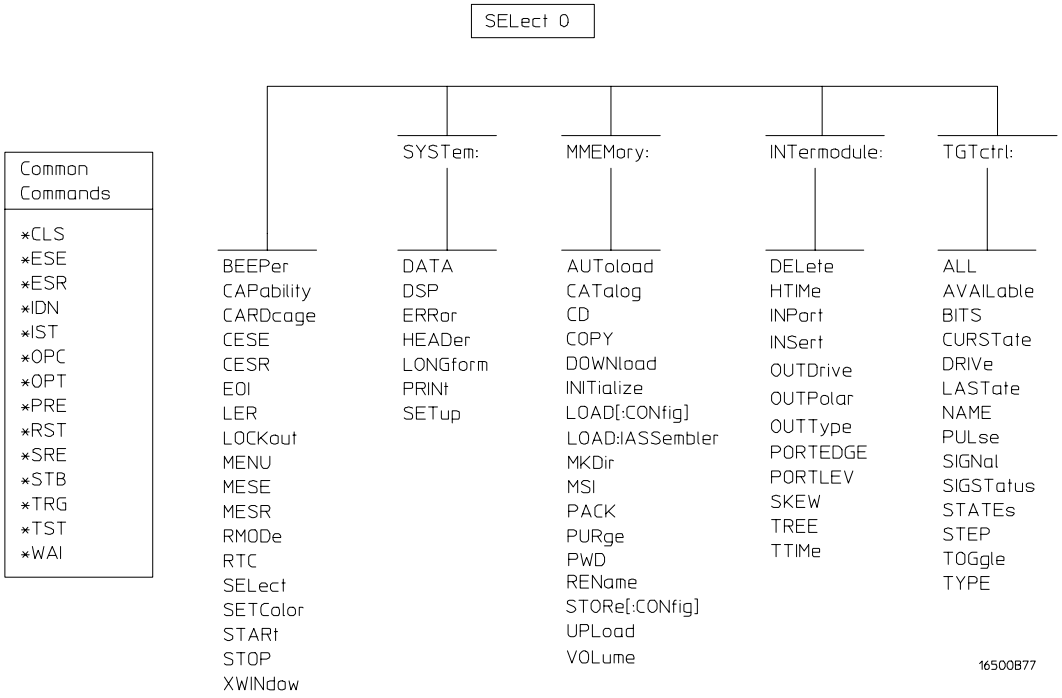
As shown in chapter 1, "Header Types," there are three types of headers. Each header has a corresponding command type. This section shows how they relate to the command tree.

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.

Figure 5-1



HP 16500C Command Tree

Tree Traversal Rules

Command headers are created by traversing down the command tree. A legal command header from the command tree in figure 5-1 would be **:MMEMORY:INITIALIZE**. This is referred to as a compound header. 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. For example, the colon preceding **MMEMORY** (**:MMEMORY**) in the above example 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. For example, the colon separating **MMEMORY** and **INITIALIZE** is the location of the parser when this compound header is parsed.

The following examples are written using HP BASIC 6.2 on a HP 9000 Series 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

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

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.

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

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

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

Example

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"
```

Command Set Organization

The command set for the HP 16500C Logic Analysis System mainframe is divided into 6 separate groups as shown in figure 5-1. The command groups are: common commands, mainframe commands, and 4 sets of subsystem commands. In addition to the command tree in figure 5-1, a command to subsystem cross-reference is shown in table 5-1.

Each of the 6 groups of commands is described in a separate 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 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 four subsystems in the mainframe. In the command tree (figure 5-1) 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 four subsystems in the HP 16500C Logic Analysis System 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).
- TGTctrl – provides access to the target control signals.

Table 5-1

Alphabetic Command Cross-Reference

Command	Subsystem	Command	Subsystem	Command	Subsystem
*CLS	Common	DATA	SYSTem	PORTEDGE	INTermodule
*ESE	Common	DELete	INTermodule	PORTLEV	INTermodule
*ESR	Common	DOWNload	MMEMory	PRINt	SYSTem
*IDN	Common	DRIVe	TGTctrl	PULSe	TGTctrl
*IST	Common	DSP	SYSTem	PURGe	MMEMory
*OPC	Common	EOI	Mainframe	PWD	MMEMory
*OPT	Common	ERRor	SYSTem	REName	MMEMory
*PRE	Common	HEADer	SYSTem	RMODe	Mainframe
*RST	Common	HTIMe	INTermodule	RTC	Mainframe
*SRE	Common	INITialize	MMEMory	SELect	Mainframe
*STB	Common	INPort	INTermodule	SETColor	Mainframe
*TRG	Common	INSert	INTermodule	SIGNal	TGTctrl
*TST	Common	LASTstate	TGTctrl	SIGStatus	TGTctrl
*WAI	Common	LER	Mainframe	SKEW	INTermodule
ALL	TGTctrl	LOAD	MMEMory	STARt	Mainframe
AUToload	MMEMory	LOCKout	Mainframe	STATes	TGTctrl
AVAILable	TGTctrl	LONGform	SYSTem	STEP	TGTctrl
BEEPer	Mainframe	MENU	Mainframe	STOP	Mainframe
BITS	TGTctrl	MESE	Mainframe	STORe	MMEMory
CAPability	Mainframe	MESR	Mainframe	SETup	SYSTem
CARDcage	Mainframe	MKDir	MMEMory	TOGGle	TGTctrl
CATalog	MMEMory	MSI	MMEMory	TREE	INTermodule
CD	MMEMory	NAME	TGTctrl	TTIMe	INTermodule
CESE	Mainframe	OUTDrive	INTermodule	TYPE	TGTctrl
CESR	Mainframe	OUTPolar	INTermodule	UPLoad	MMEMory
COPY	MMEMory	OUTType	INTermodule	VOLume	MMEMory
CURSTate	TGTctrl	PACK	MMEMory		

Program Examples

The program examples in chapter 15, "Programming Examples," were written on an HP 9000 Series 300 controller using the HP BASIC 6.2 language. The programs always assume a generic address for the HP 16500C Logic Analysis System of 707. The shorter examples given in the reference sections use a generic address 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 logic analyzer's Timing Waveform Delay to 100 ms.

Keywords in long form, numbers using the decimal format.

```
OUTPUT XXX;":SELECT 2:MACHINE1:TWAVEFORM:DELAY .1"
```

Keywords in short form, numbers using an exponential format.

```
OUTPUT XXX;":SEL 2:MACH1:TWAV:DEL 1E-1"
```

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

```
OUTPUT XXX;":sel 2:mach1:twav:del 100ms"
```

In these examples, the colon shown as the first character of the command is optional on the HP 16500C Logic Analysis System. The space between DELay and the argument is required.

6

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 Analysis System. 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 16500C Logic Analysis System 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-232-C program messages and response messages for the HP 16500C Logic Analysis System are structured very similar to those described by IEEE 488.2. In most cases, the same structure shown in this chapter for IEEE 488.2 will also work for RS-232-C. Because of this, no additional information has been included for RS-232-C.

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 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 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 messages and response messages. These messages serve as the containers into which sets of program commands or instrument responses are placed. Program messages are sent by the controller to the instrument, and response messages 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. An important 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 messages and response messages. This means that the controller should always terminate a program message before attempting to read a response. The instrument will terminate response messages 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 noted in chapter 1, "Multiple Queries," 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.

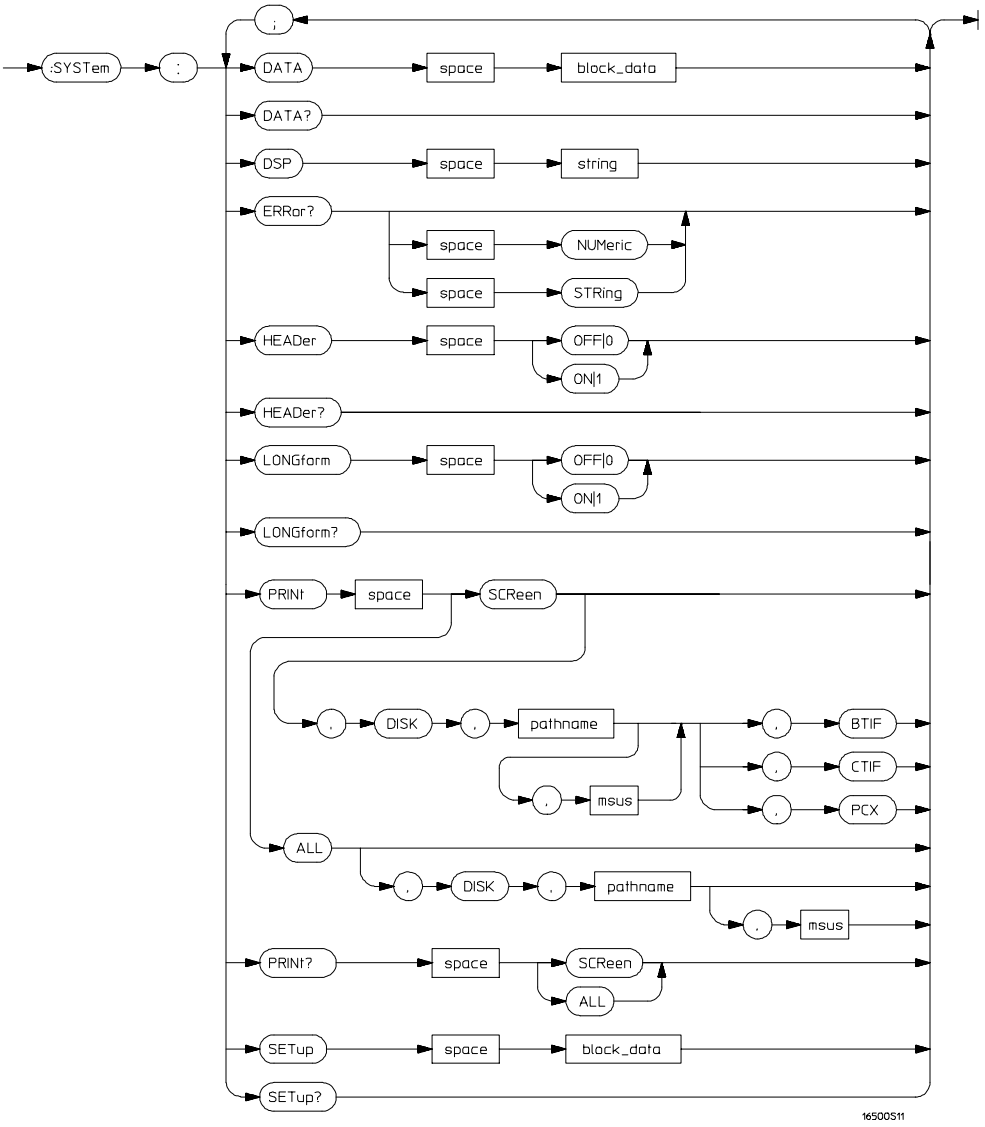
Query Error A query error will be reported if the proper protocol for reading a query is not followed. This includes the interrupted and unterminated conditions described in the following paragraphs.

Syntax Diagrams

The example syntax diagram in this chapter is 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 6-1



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.

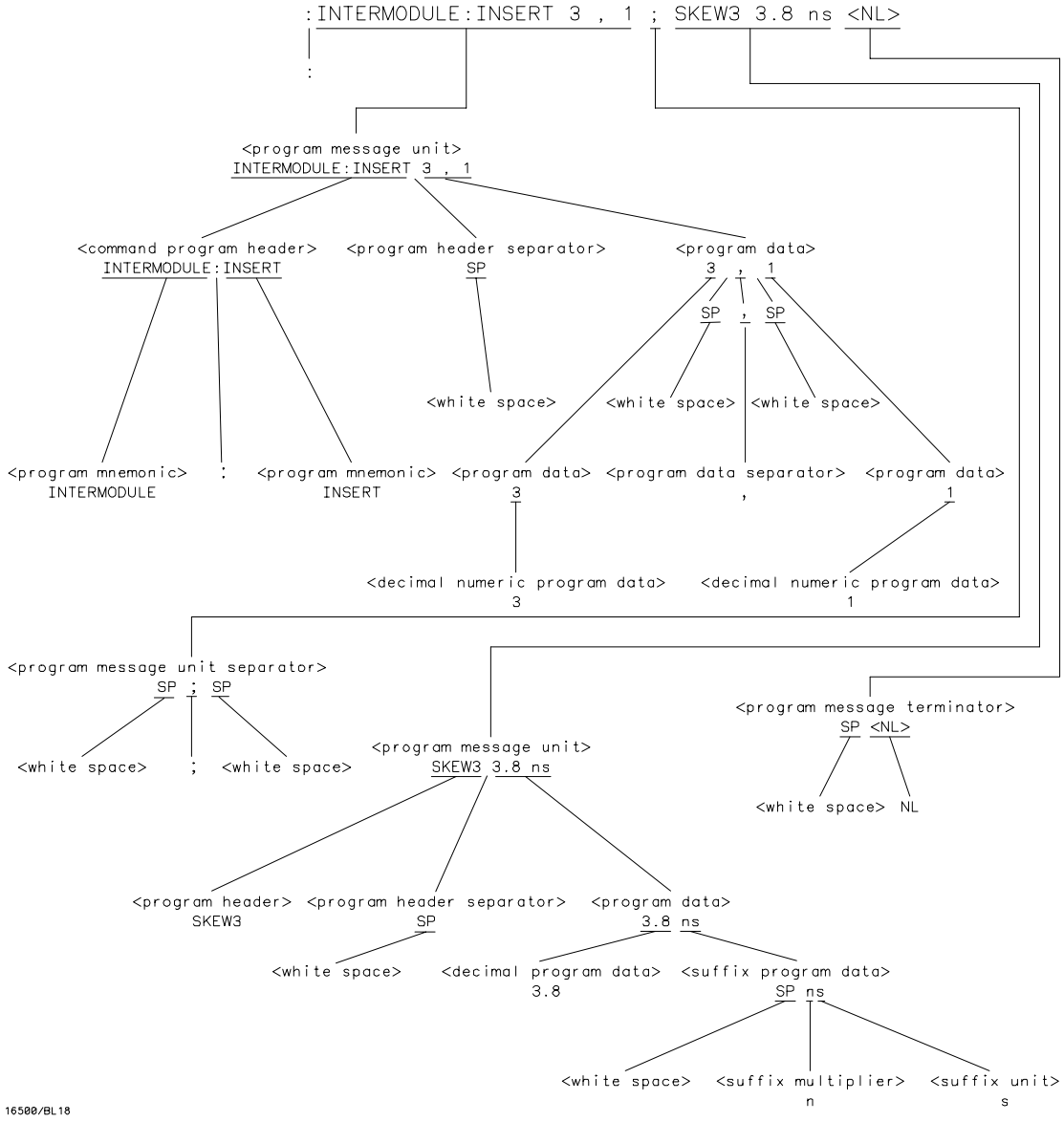
IEEE 488.2 defines the blocks used to build messages which are sent to the instrument. A whole string of commands can therefore be broken up into individual components.

Figure 6-1 is an example syntax diagram and figure 6-2 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 units 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.
- Multiple data parameters are separated by a comma.
- The first data parameter is separated from the header with one or more spaces.
- The header **SYSTEM:LONGFORM OFF** 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.

See Also Chapter 5, "Programming and Documentation Conventions"

Figure 6-2



<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 6-1. They are used in conjunction with suffix units, shown in table 6-2.

Table 6-1 **<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 6-2.

Table 6-2 **<suffix unit>**

Suffix	Referenced Unit
V	Volt
S	Second

Example To specify 3 ns, you might enter `3NS` or `3E-9 S` in your program.

7

Status Reporting

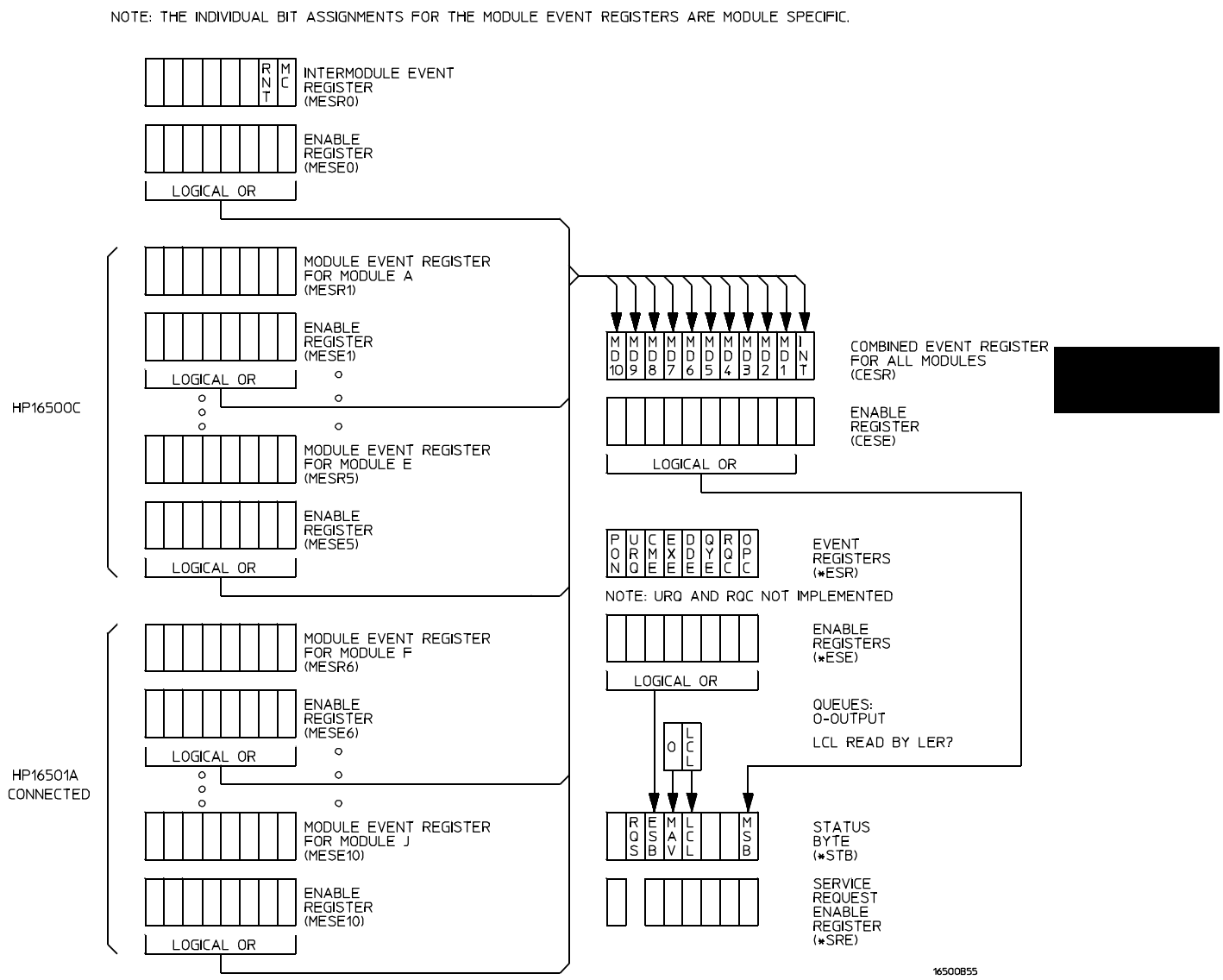
Introduction

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 16500C Logic Analysis System. Also in this chapter is a sample set of steps you might use to perform a serial poll over HP-IB.

The status reporting features available over the bus are the serial and parallel polls. 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 7-1



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 7-1 and in chapter 9, "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 16500C Logic Analysis System).

PON - power on

Indicates power has been turned on.

URQ - user request

Always returns a 0 from the HP 16500C Logic Analysis System.

CME - command error

Indicates whether the parser detected an 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.

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? STRING.

RQC - request control

Always returns a 0 from the HP 16500C Logic Analysis System.

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 module 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

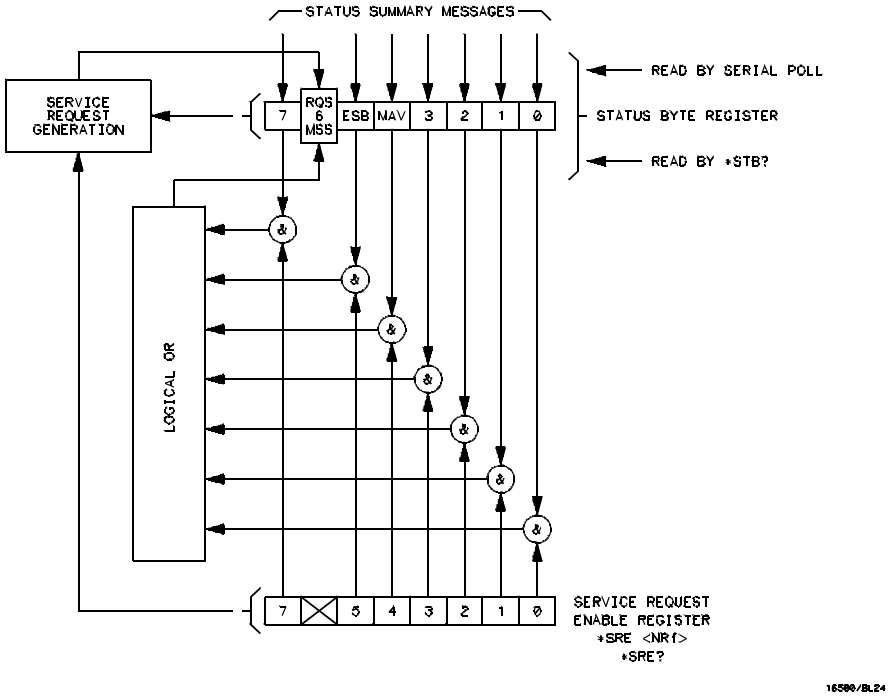
```
OUTPUT XXX;"*SRE 32 ; *ESE 1"  !enables an OPC service
request
```

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 7-2



Service Request Enabling

Serial Poll

The HP 16500C Logic Analysis System 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: the Logic Analysis System 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 Analysis System 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 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 Status Byte Register of the HP 16500C Logic Analysis System 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.

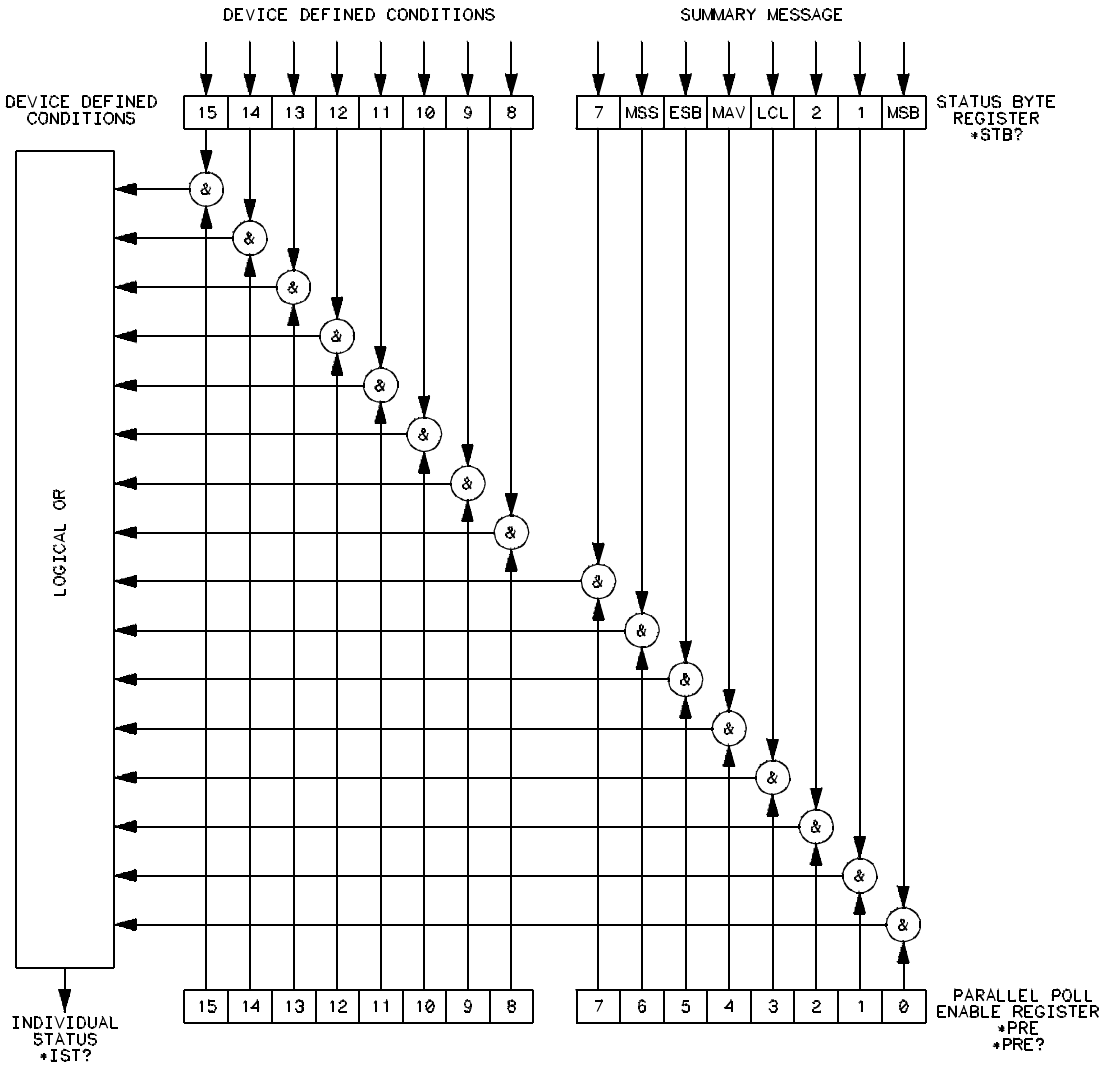
Parallel Poll

Parallel poll is a controller-initiated operation which is used to obtain information from several devices simultaneously. When a controller initiates a Parallel Poll, each device returns a Status Bit via one of the DIO data lines. Device DIO assignments are made by the controller using the PPC (Parallel Poll Configure) sequence. Devices respond either individually, each on a separate DIO line; collectively on a single DIO line; or any combination of these two ways. When responding collectively, the result is a logical AND (True High) or logical OR (True Low) of the groups of status bits.

Figure 7-3 shows the Parallel Poll Data Structure. The summary bit is sent in response to a Parallel Poll. This summary bit is the "IST" (individual status) local message.

The Parallel Poll Enable Register determines which events are summarized in the IST. The ***PRE** command is used to write to the enable register and the ***PRE?** query is used to read the register. The ***IST?** query can be used to read the IST without doing a parallel poll.

Figure 7-3



Parallel Poll Data Structure

Polling HP-IB Devices

Parallel poll is the fastest means of gathering device status when several devices are connected to the bus. Each device (with this capability) can be programmed to respond with one bit of status when parallel polled. This makes it possible to obtain the status of several devices in one operation. If a device responds affirmatively to a parallel poll, more information about its specific status can be obtained by conducting a serial poll of the device.

Configuring Parallel Poll Responses

Certain devices, including the HP 16500C Logic Analysis System, can be remotely programmed by a controller to respond to a parallel poll. A device which is currently configured for a parallel poll responds to the poll by placing its current status on one of the bus data lines. The response and the data-bit number can then be programmed by the PPC (parallel poll configure) statement. No multiple listeners can be specified in this statement. If more than one device is to respond on a single bit, each device must be configured with a separate PPC statement.

Example

```
ASSIGN @Device TO 707
PPOLL CONFIGURE @Device;Mask
```

The value of Mask (any numeric expression can be specified) is first rounded and then used to configure the device's parallel response. The least significant 3 bits (bits 0 through 2) of the expression are used to determine which data line the device is to respond on (place its status on). Bit 3 specifies the "true" state of the parallel poll response bit of the device. A value of 0 implies that the device's response is 0 when its status bit message is true.

Example

The following statement configures the device at address 07 on the interface select code 7 to respond by placing a 0 on bit 4 when its status response is "true."

```
PPOLL CONFIGURE 707;4
```

Conducting a Parallel Poll

The PPOLL (Parallel Poll) function returns a single byte containing up to 8 status bit messages for all devices on the bus capable of responding to the poll. Each bit returned by the function corresponds to the status bit of the device(s) configured to respond to the parallel poll (one or more devices can respond on a single line). The PPOLL function can only be executed by the controller. It is initiated by the simultaneous assertion of ATN and EOI.

Example

```
Response = PPOLL(7)
```

Disabling Parallel Poll Responses

The PPU (Parallel Poll Unconfigure) statement gives the controller the capability of disabling the parallel poll responses of one or more devices on the bus.

Example

The following statement disables device 5 only:

```
PPOLL UNCONFIGURE 705
```

This statement disables all devices on interface select code 8 from responding to a parallel poll:

```
PPOLL UNCONFIGURE 8
```

If no primary address is specified, all bus devices are disabled from responding to a parallel poll. If a primary address is specified, only the specified devices (which have the parallel poll configure capability) are disabled.

HP-IB Commands

The following paragraphs describe actual HP-IB commands which can be used to perform the functions of the BASIC commands shown in the previous examples.

Parallel Poll Unconfigure Command

The parallel poll unconfigure command (PPU) resets all parallel poll devices to the idle state (unable to respond to a parallel poll).

Parallel Poll Configure Command

The parallel poll configure command (PPC) causes the addressed listener to be configured according to the parallel poll enable secondary command PPE.

Parallel Poll Enable Command

The parallel poll enable secondary command (PPE) configures the devices which have received the PPC command to respond to a parallel poll on a particular HP-IB DIO line with a particular level.

Parallel Poll Disable Command

The parallel poll disable secondary command (PPD) disables the devices which have received the PPC command from responding to the parallel poll.

Table 7-1 Parallel Poll Commands

Command	Mnemonic	Decimal Code	ASCII/ISO Character
Parallel Poll Unconfigure (Multiline Command)	PPU	21	NAK
Parallel Poll Configure (Addressed Command)	PPC	05	ENQ
Parallel Poll Enable (Secondary Command)	PPE	96-111	I-O
Parallel Poll Disable (Secondary Command)	PPD	112	P

Error Messages

Introduction

This chapter lists the error messages that relate to the HP 16500C Logic Analysis System.

Device Dependent Errors

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

Part 2

9 Common Commands 9-1
10 Mainframe Commands 10-1
11 SYSTem Subsystem 11-1
12 MMEMory Subsystem 12-1
13 INTermodule Subsystem 13-1
14 TGTctrl Subsystem 14-1

Commands

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 9-1 and table 9-1 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. The common commands are:

- *CLS
- *ESE
- *ESR
- *IDN
- *IST
- *OPC
- *OPT
- *PRE
- *RST
- *SRE
- *STB
- *TRG
- *TST
- *WAI

Common commands can be received and processed by the HP 16500C Logic Analysis System, 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 system will remain in the selected subsystem.

Example

If the program message in this example is received by the system, it will initialize the disk and store the file and clear the status information. This is not 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 re-enter the memory subsystem and store the file.

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

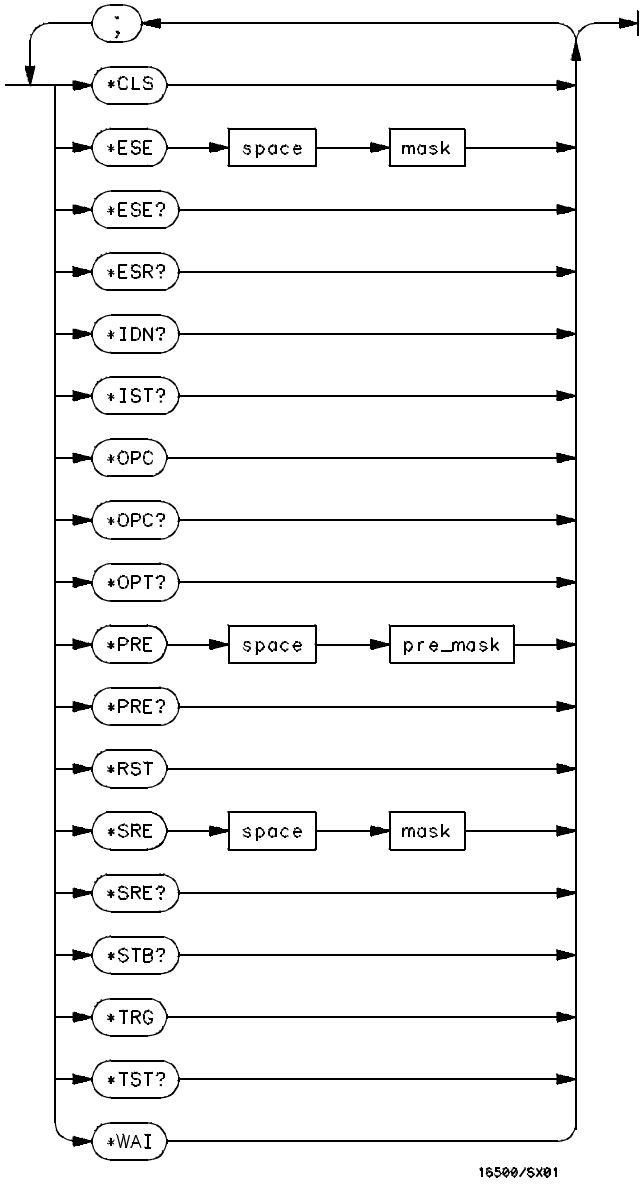
Status Registers

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.

See Also

Chapter 7, "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 9-1



Common Commands Syntax Diagram

Table 9-1 **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.

Example

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

See Also

Refer to chapter 7, "Status Reporting," for a complete discussion of status.

*ESE (Event Status Enable)

Command

```
*ESE <mask>
```

The *ESE command sets the Standard Event Status Enable Register bits. The Standard Event Status Enable Register contains a bit to enable the status indicators detailed in table 9-2. A 1 in any bit position of the Standard Event Status Enable Register enables the corresponding status in the Standard Event Status Register.

`<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?"
```

See Also Refer to chapter 7, "Status Reporting" for a complete discussion of status.

Table 9-2 **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 9-3 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 9-3 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 error has been detected
4	16	EXE	0 = no execution errors 1 = an execution error has been detected
3	8	DDE	0 = no device dependent errors 1 = a device dependent error has been detected
2	4	QYE	0 = no query errors 1 = a query error has been detected
1	2	RQC	0 = request control - not used - always 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,16500C,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,16500C,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 9-2 shows the *IST data structure.

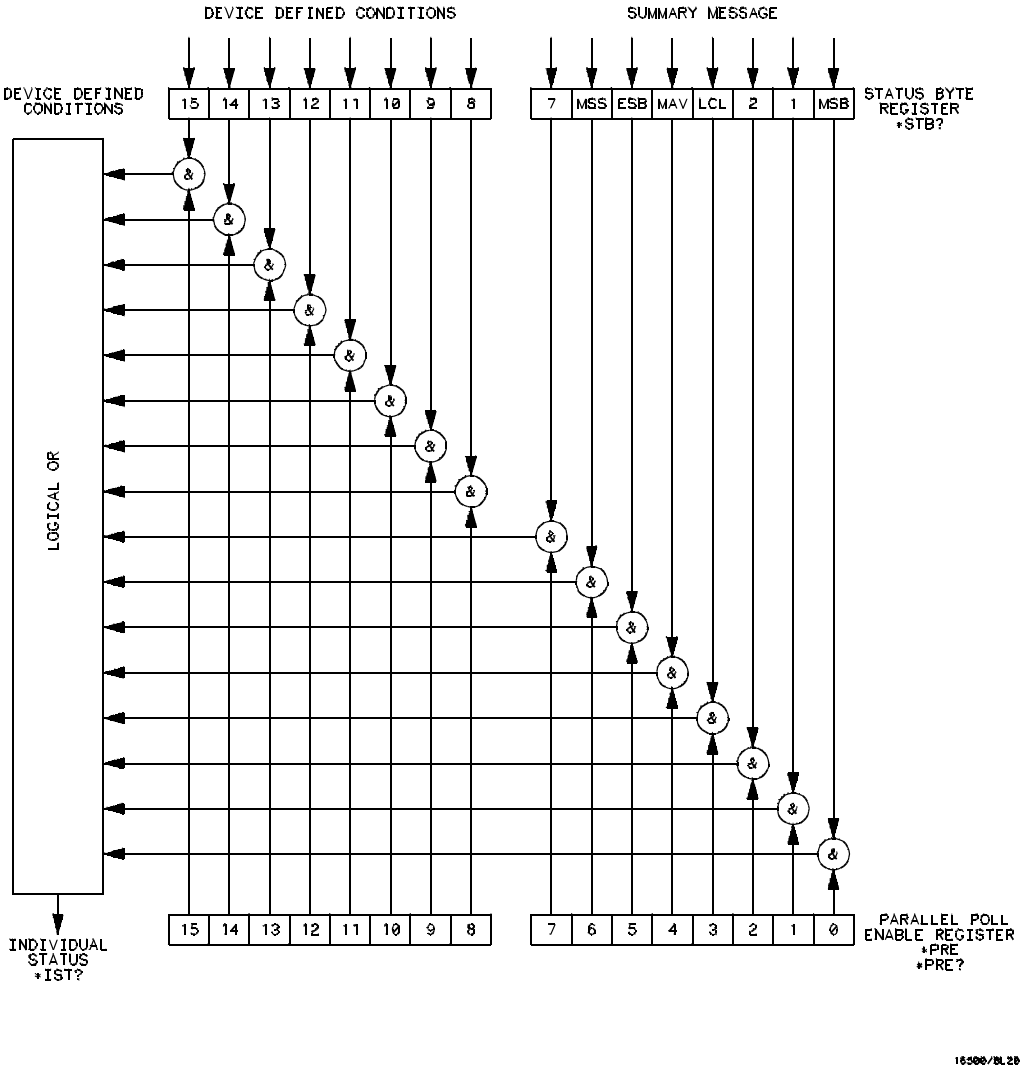
Returned Format `<id><NL>`

`<id>` {0|1} 0 indicates the IST local message is true; 1 indicates the IST local message is false.

Example

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

Figure 9-2



***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 16500C 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 16500C. 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 16500C mainframe. When an HP 16501A Expansion frame is connected, there will be ten parameters after the INTERMODULE for modules in slots A through J. A zero in any of the last eight parameters indicates that the corresponding software is not currently installed.

Returned Format

```
{SYSTEM},{<option>|0},{<option>|0},{INTERMODULE|0},{<module>|0}
,{<module>|0},{<module>|0},{<module>|0},{<module>|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 9-4 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 allows the HP 16500C 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?"
```

See Also Chapter 7, "Parallel Poll," for more information on conducting a parallel poll.

Table 9-4 **HP 16500C 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 16500C. The HP 16500C will accept this command, but the command has no affect on the system.

The *RST command is generally used to place the system in a predefined state. Because the HP 16500C allows you to store predefined configuration files for individual modules, or for the entire system, resetting the system can be accomplished by simply loading the appropriate configuration file.

See Also For more information, refer to chapter 12, "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 9-5 for the bits in the Service Request Enable Register and what they mask.

`<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?"
```

See Also

Refer to Chapter 7, "Status Reporting," for a complete discussion of status.

Table 9-5 **HP 16500C 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, 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 9-6 for the meaning of the bits in the status byte.

Returned Format `<value><NL>`

`<value>` An integer from 0 through 255

Example

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

See Also Refer to chapter 7, "Status Reporting" for a complete discussion of status.

Table 9-6 The Status Byte Register

Bit Position	Bit Weight	Bit Name	Condition
7	128		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 9-7 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 9-7 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	Interrupt 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 16500C are **STARt** and **STOP**.

Example

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

Mainframe Commands

Introduction

Mainframe commands control the basic operation of the instrument for both the HP 16500C mainframe alone or with the HP 16501A expansion frame connected. Mainframe commands can be called at anytime, and from any module. The only difference in mainframe commands with an HP 16501A connected is the number of slots and modules. These differences will be noted in the affected command descriptions.

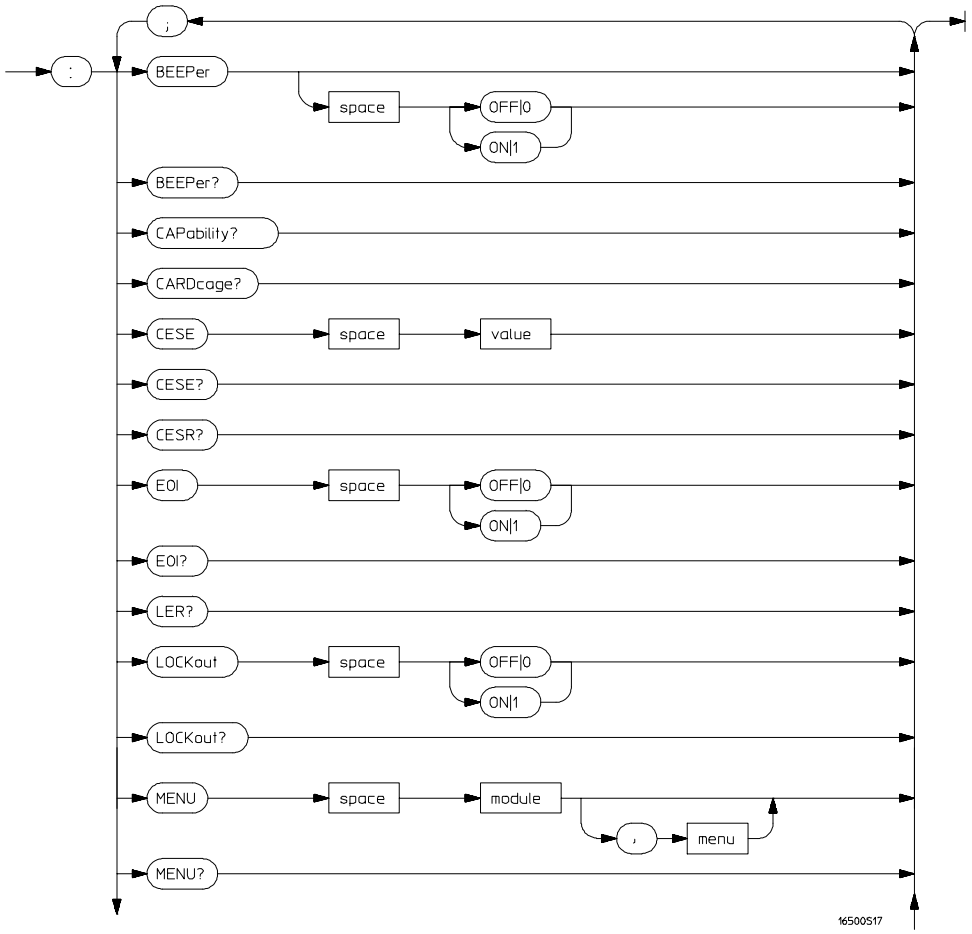
The main difference between an HP 16500C alone and an HP 16500C with the HP 16501A connected is how you specify the **SELECT** command. The HP 16500C alone has only five slots; therefore, if you specify 6 through 10 for the **SELECT** command in your program, the command parser will take no action.

This chapter contains the mainframe commands with a syntax example for each command. Each syntax example contains the parameters for the HP 16500C/16501A. Refer to figure 10-1 and table 10-1 for the syntax diagram of the Mainframe commands.

The mainframe commands are:

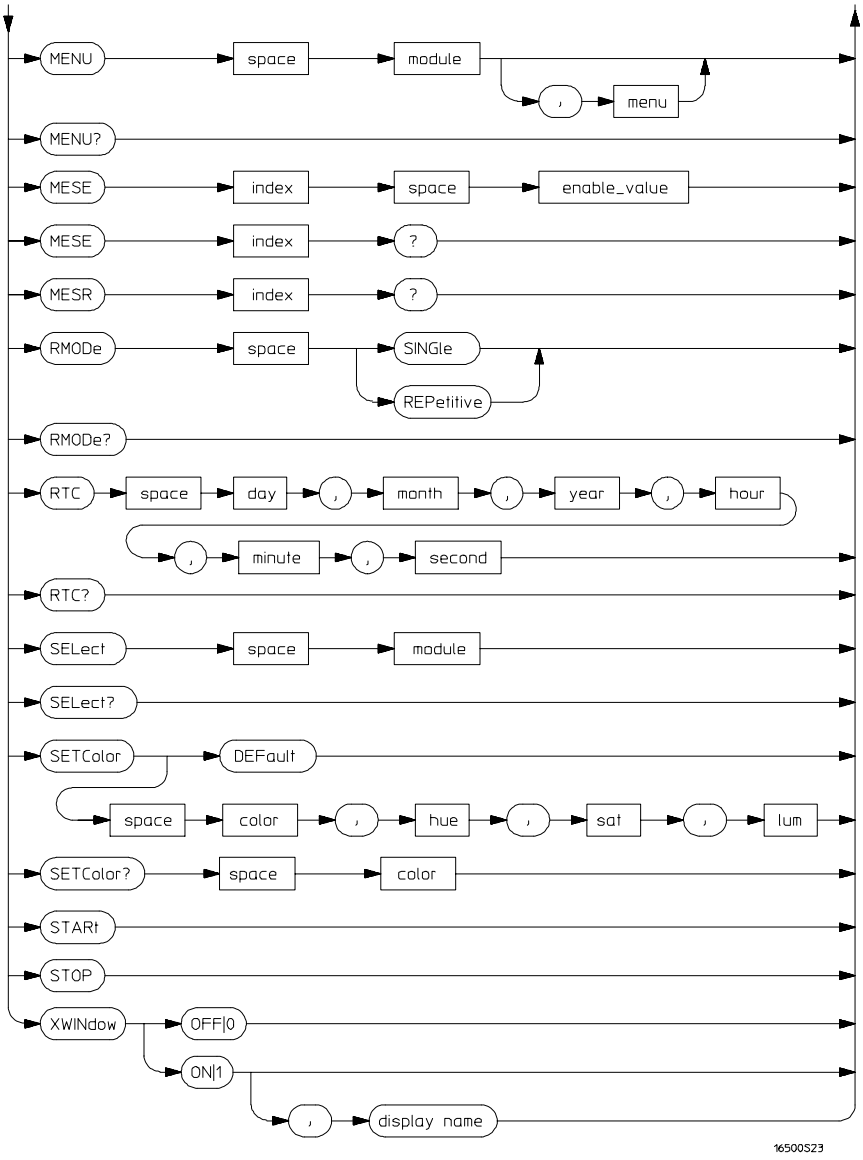
- BEEPer
- CAPability
- CARDcage
- CESE
- CESR
- EOI
- LER
- LOCKout
- MENU
- MESE
- MESR
- RMODe
- RTC
- SELect
- SETColor
- STARt
- STOP
- XWINdow

Figure 10-1



Mainframe Commands Syntax Diagram

Figure 10-1 (continued)



Mainframe Commands Syntax Diagram (continued)

Table 10-1 Mainframe Parameter Values

Parameter	Values
value	An integer from 0 to 65535
module	An integer from –2 through 5 for an HP 16500C alone or from –2 through 10 with an HP 16501A connected
menu	An integer
enable_value	An integer from 0 to 255
index	An integer from 0 to 5 for an HP 16500C alone or from 0 to 10 with an HP 16501A connected
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
sat	An integer from 0 to 100
lum	An integer from 0 to 100
display name	A string containing an IP Address and a display name, for example, "12.3.19.1:0.0"

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 IEEE 488.1 "Interface Requirements for Devices" capability sets implemented in the device.

Table 10-2 lists the capability sets implemented in the HP 16500C.

Returned Format

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

Example

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

Table 10-2 HP 16500C 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 identify the modules that are installed in the mainframe. For an HP 16500C alone, the first five numbers returned are the card identification numbers (–1 means no card is in the slot). The remaining five numbers returned indicate the module assignment (that is, the slot containing the master card of the module) for each card. For single-card modules, the module assignment is the same as the card's slot. The possible values for the module assignment are 0, 1, 2, 3, 4, and 5 where 0 indicates an empty slot or the module software is not recognized or not loaded. 1...5 indicates the number of the slot in which the master card for this card is located.

When an HP 16501A is connected, the first ten numbers returned are the card identification numbers (–1 means no card is in the slot). The remaining ten numbers returned indicate the module assignment for each card. The possible values for the module assignment are 0 through 10 where 0 indicates an empty slot or the module software is not recognized or not loaded. 1...10 indicates the number of the slot in which the master card for this card is located.

Table 10-3 lists the card identification numbers and their associated cards.

Returned Format

```
[:CARDcage]
<ID>,<ID>,<ID>,<ID>,<ID>,[<ID>,<ID>,<ID>,<ID>,<ID>,]
<assign>,<assign>,<assign>,<assign>,<assign>
[,<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?"
```

Table 10-3 Card Identification Numbers

Id Number	Card
1	HP 16515A 1GHz Timing Master Card
2	HP 16516A 1GHz Timing Expansion Card
4	HP 16517A 4GHz Timing/1GHz State Analyzer Master Card
5	HP 16518A 4GHz Timing/1GHz State Analyzer Expansion Card
11	HP 16530A 400 MSa/s Oscilloscope Timebase Card
12	HP 16531A Oscilloscope Acquisition Card
13	HP 16532A 1GSa/s Oscilloscope Card
14	HP 16533A or HP 16534A 32K GSa/s Oscilloscope Card
15	HP 16535A MultiProbe 2-Output Module
21	HP 16520A Pattern Generator Master Card
22	HP 16521A Pattern Generator Expansion Card
24	HP 16522A 200MHz Pattern Generator Expansion Card
25	HP 16522A 200MHz Pattern Generator Master Card
30	HP 16511B Logic Analyzer Card
31	HP 16510A or B Logic Analyzer Card
32	HP 16550A 100/500 MHz Logic Analyzer Master Card
33	HP 16550A 100/500 MHz Logic Analyzer Expansion Card
34	HP 16554, 16555, or 16556 Logic Analyzer Master Card
35	HP 16554, 16555, or 16556 Logic Analyzer Expansion Card
40	HP 16540A 100/100 MHz Logic Analyzer Master Card
41	HP 16541A 100/100 MHz Logic Analyzer Expansion Card
42	HP 16542A 2 MB Acquisition Logic Analyzer Master Card
43	HP 16542A 2 MB Acquisition Logic Analyzer Expansion Card

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 16500C. Table 10-4 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 10-4 **HP 16500C Combined Event Status Enable Register**

Bit	Weight	Enables
11-15		not used
10	1024	Module in slot J
9	512	Module in slot I
8	256	Module in slot H
7	128	Module in slot G
6	64	Module in slot F
5	32	Module in slot E
4	16	Module in slot D
3	8	Module in slot C
2	4	Module in slot B
1	2	Module in slot A
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 16500C System. Table 10-5 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 10-5 **HP 16500C Combined Event Status Register**

Bit	Bit Weight	Bit Name	Condition
11-15			not used
10	1024	Module J	0 = No new status 1 = Status to report
9	512	Module I	0 = No new status 1 = Status to report
8	256	Module H	0 = No new status 1 = Status to report
7	128	Module G	0 = No new status 1 = Status to report
6	64	Module F	0 = No new status 1 = Status to report
5	32	Module E	0 = No new status 1 = Status to report
4	16	Module D	0 = No new status 1 = Status to report
3	8	Module C	0 = No new status 1 = Status to report
2	4	Module B	0 = No new status 1 = Status to report
1	2	Module A	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 the last byte of a reply from the HP 16500C is to be sent with the EOI bus control line set true. If EOI is turned off, the logic analyzer responses will not be IEEE 488.2 compliant.

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 10-6 lists the module parameters. The mainframe menus and parameters are listed in table 10-7.

`<module>` Selects module or system. An integer from –2 through 5 for HP 16500C only or an integer from –2 through 10 with an HP 16501A connected.

`<menu>` Selects menu (integer)

Example

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

See Also

Programmer's Guide for specific module for the module's `<menu>` values.

Table 10-6 **`<module>` values**

Parameter	Menu
0	System/Intermodule
1	Module in slot A
2	Module in slot B
3	Module in slot C
4	Module in slot D
5	Module in slot E
–1	Software option 1
–2	Software option 2
Available when an HP 16501A is connected:	
6	Module in slot F
7	Module in slot G
8	Module in slot H
9	Module in slot I
10	Module in slot J

Table 10-7 **System Menu Values**

Menu Command Parameters	Menu
MENU 0,0	System Configuration menu
MENU 0,1	Hard disk menu
MENU 0,2	Flexible disk menu
MENU 0,3	Utilities menu
MENU 0,4	Test menu
MENU 0,5	Intermodule menu

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 16500C alone, the <N> index 0 through 5 refers to system and modules 1 through 5 respectively. With an HP 16501A connected, the <N> index 6 through 10 refers to modules 6 through 10 respectively. Table 10-8 lists the Module Event Status Enable register bits, bit weights, and what each bit masks for the mainframe.

`<N>` An integer 0 through 10

`<enable_value>` An integer from 0 through 255

Example

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

Query

```
:MESE<N>?
```

The query returns the current setting. Table 10-8 lists the Module Event Status Enable register bits, bit weights, and what each bit masks for the mainframe.

Returned Format

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

Example

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

Table 10-8 HP 16500C 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

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 16500C alone, the <N> index 0 through 5 refers to system and modules 1 through 5 respectively. With an HP 16501A connected, the <N> index 6 through 10 refers to modules 6 through 10 respectively.

Refer to table 10-9 for information about the Module Event Status Register 0 bits and their bit weights.

See Also

MESR in the *Programmer's Guide* for a specific module for the interpretation of that module's Event Status Register.

Returned Format

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

<N> An integer 0 through 1.

<enable_value> An integer from 0 through 255

Example

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

Table 10-9 HP 16500C Mainframe Module Event Status Register (<N>=0)

Bit	Bit Weight	Bit Name	Condition
7	128		not used
6	64		not used
5	32		not used
4	16		not used
3	8		not used
2	4		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

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>`

The real-time clock command allows you to set the real-time clock to the current date and time.

`<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. The appropriate module (or system) must be selected before any module (or system) specific commands can be sent. SELECT 0 selects System, SELECT 1 through 5 selects modules A through E in an HP 16500C only. SELECT 1 through 10 selects modules A through J when an HP 16501A is connected. –1 and –2 selects software options 1 and 2 respectively. The query returns the current module selection.

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 10-2 shows the command tree for the SELect command.

`<module>` Selects module or system. An integer from –2 through 5 for HP 16500C only or an integer from –2 through 10 with an HP 16501A connected.

Example

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

Query

```
:SELect?
```

The SELect? query returns the current module selection.

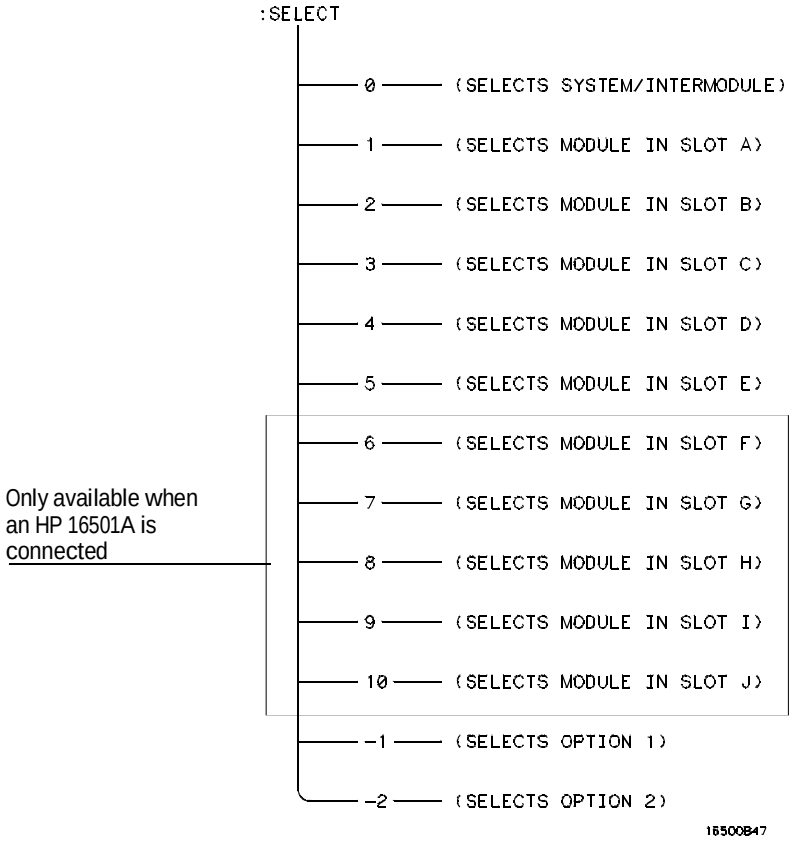
Returned Format

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

Example

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

Figure 10-2



Select Command Tree

SETColor

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

The SETColor command is used to change one of the color selections on the CRT, or to return to the default screen colors. Four parameters are sent with the command to change a color:

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

`<color>` An integer from 1 to 7

`<hue>` An integer from 0 to 100

`<sat>` An integer from 0 to 100

`<lum>` An integer from 0 to 100

Color Number 0 cannot be changed.

Example

```
OUTPUT XXX;":SETCOLOR 3,60,100,60"
OUTPUT XXX;":SETC DEFAULT"
```

Query `:SETColor? <color>`

The SETColor query returns the hue, saturation, and luminosity values for a specified color.

Returned Format `[:SETColor] <color>,<hue>,<sat>,<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"
```

XWINdow

Command

```
:XWINdow {{OFF|0} | {ON|1}[,<display>]}
```

The XWINdow command opens or closes a window on an X Window display server, that is, a networked workstation or personal computer. The XWINdow ON command opens a window. If no display is specified, the display already stored in the HP 16500C X Window Settings menu is used. If a display is specified, that one is used. The specified display also is stored in non-volatile memory in the HP 16500C.

`<display>` A string containing an Internet (IP) Address optionally followed by a display and screen specifier. For example,

```
"12.3.47.11"
```

or

```
"12.3.47.11:0.0"
```

Example

To open a window, specifying and storing the display name:

```
OUTPUT XXX;":XWINDOW ON,'12.3.47.11'"
```

To open a window, using the stored display name:

```
OUTPUT XXX;":XWIN ON"
```

To close the X Window:

```
OUTPUT XXX;":XWINDOW OFF"
```

If you have trouble displaying an X Window, check that your server permits windows from the HP 16500C to be displayed. On UNIX systems, the command is "`xhost +<16500 IP>`". See your network documentation for more details.

11

SYSTem Subsystem

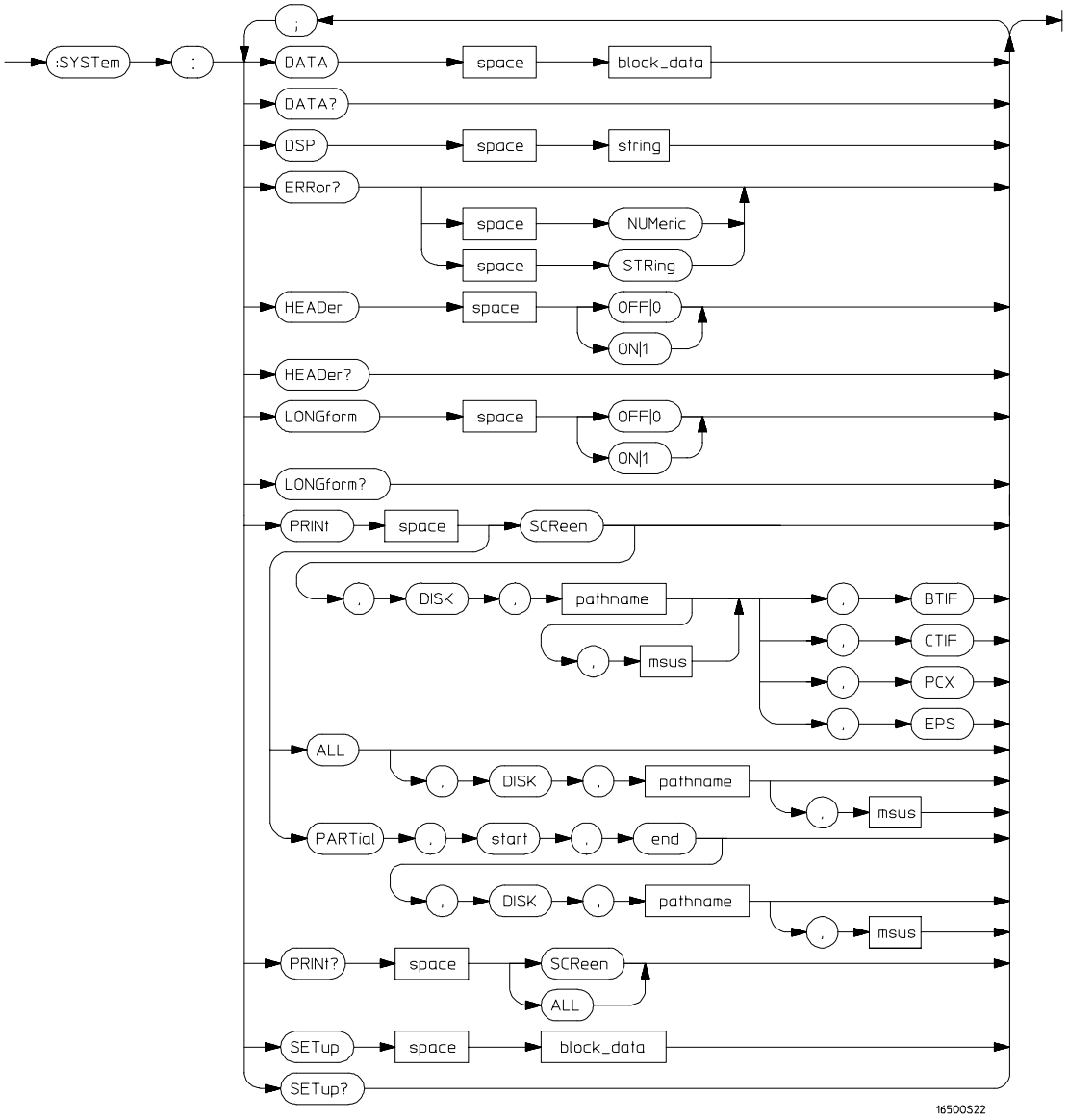
Introduction

SYSTem subsystem commands control functions that are common to the entire logic analysis system, including formatting query responses and enabling reading and writing to the advisory line on the display of the HP 16500C mainframe.

Refer to figure 11-1 and table 11-1 for the SYSTem Subsystem commands syntax diagram. The SYSTem Subsystem commands are:

- DATA
- DSP
- ERRor
- HEADer
- LONGform
- PRINt
- SETup

Figure 11-1



System Subsystem Commands Syntax Diagram

Table 11-1 SYSTem Parameter Values

Parameter	Values
block_data	Data in IEEE 488.2 format.
string	A string of up to 68 alphanumeric characters.
pathname	A string of up to 10 alphanumeric characters for LIF in the following form: **NNNNNNNNNN** or A string of up to 64 alphanumeric characters for DOS in one of the following forms: **NNNNNNNN . NNN** when the file resides in the present working directory or \NAME_DIR\FILENAME when the file does not reside in the present working directory

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 the logic analysis system
- Processing data later in the logic analysis system
- 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; however, the mainframe by itself does not have acquired data. Therefore, the DATA command and query are described in detail in the respective module *Programmer's Guides*. Because the capabilities of the DATA command and query vary for individual modules, a complete chapter is dedicated to the DATA command and query in each of the module *Programmer's Guides*. The dedicated chapter is called "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 of the individual module *Programmer's Guides*.

`<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 selected module when the last acquisition was performed. Any changes made since then through either front-panel operations or programming commands do not affect the stored data. Since the mainframe does not acquire data, refer to the appropriate module *Programmer's Guide* for more details.

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

Example See the *Programmer's Guide* for the selected module 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 is returned in the following form:

```
<error_number>,<error message string>
```

A complete list of error messages for the HP 16500C logic analysis system is shown in chapter 8, "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

Example

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 long form variable, which tells the instrument how to format query responses. If the LONGform command is set to OFF, command headers and arguments are sent from the instrument in the abbreviated form. If 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 long form or short form 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

Commands

```
:SYSTem:PRINt ALL[,DISK, <pathname>[,<msus>]]
:SYSTem:PRINt PARTial,<start>,<end>
[,DISK, <pathname>[,<msus>]]
:SYSTem:PRINt SCReen[,DISK, <pathname> [,<msus>],
{BTIF|CTIF|PCX|EPS}]
```

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. PRINt ALL is only available in menus with an ASCII representation.

PRINt SCReen allows you to specify a graphics type. The BTIF option formats the screen data in black-and-white TIF. The CTIF and PCX options format the data in color TIF and color PCX respectively. EPS specifies Encapsulated PostScript format.

If a file name extension is not specified in the command, the correct extension will be appended to the file name automatically. The file name extension is TIF for both BTIF and CTIF options.

PRINT PARTial is valid in certain listing menus. It allows you to specify a starting and ending state number so you can print just a portion of the listing to the printer or to a disk file.

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

NNNNNNNNNN
or
A string of up to 64 alphanumeric characters for DOS in one of the following forms:

NNNNNNNN.NNN when the file resides in the present working directory
or
\NAME_DIR\FILENAME when the files does not reside in the present working directory

`<msus>` Mass Storage Unit specifier. **INTernal0** for the hard disk drive and **INTernal**1 for the flexible disk drive.

`<start>` `<end>` An integer specifying a state number.

Example

This instruction prints the screen to the printer:

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

This instruction prints the entire state listing to a file named STATE:

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

This instruction prints part of a listing to disk:

```
OUTPUT XXX;":SYSTEM:PRINT PARTIAL,-9,30,DISK, 'LISTING',INT0"
```

This instruction prints a black-and-white TIF file to the hard drive:

```
OUTPUT XXX;":SYSTEM:PRINT SCREEN, DISK, 'PICTURE', INT0, BTIF"
```

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 16500C Logic Analysis System User's Reference*.

Example

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

SETup

Command `:SYSTem:SETup <block_data>`

The :SYSTem:SETup command configures the logic analysis system as defined by the block data sent by the controller. This chapter describes briefly the syntax of the Setup command and query for the mainframe. Because of the capabilities and importance of the Setup command and query for individual modules, a complete chapter is dedicated to it in each of the module *Programmer's Guides*. The dedicated chapter is called "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 the "DATA and SETup Commands" chapter of the module *Programmer's Guides*.

`<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. 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 the *Programmer's Guide* for the selected module for an example.

MMEMory Subsystem

Introduction

The MMEMory (mass memory) subsystem commands provide access to both the hard and flexible disk drives. The HP 16500C Logic Analysis System supports the DOS (Disk Operating System) format on the hard drive and both DOS and LIF (Logical Information Format) on the flexible drive.

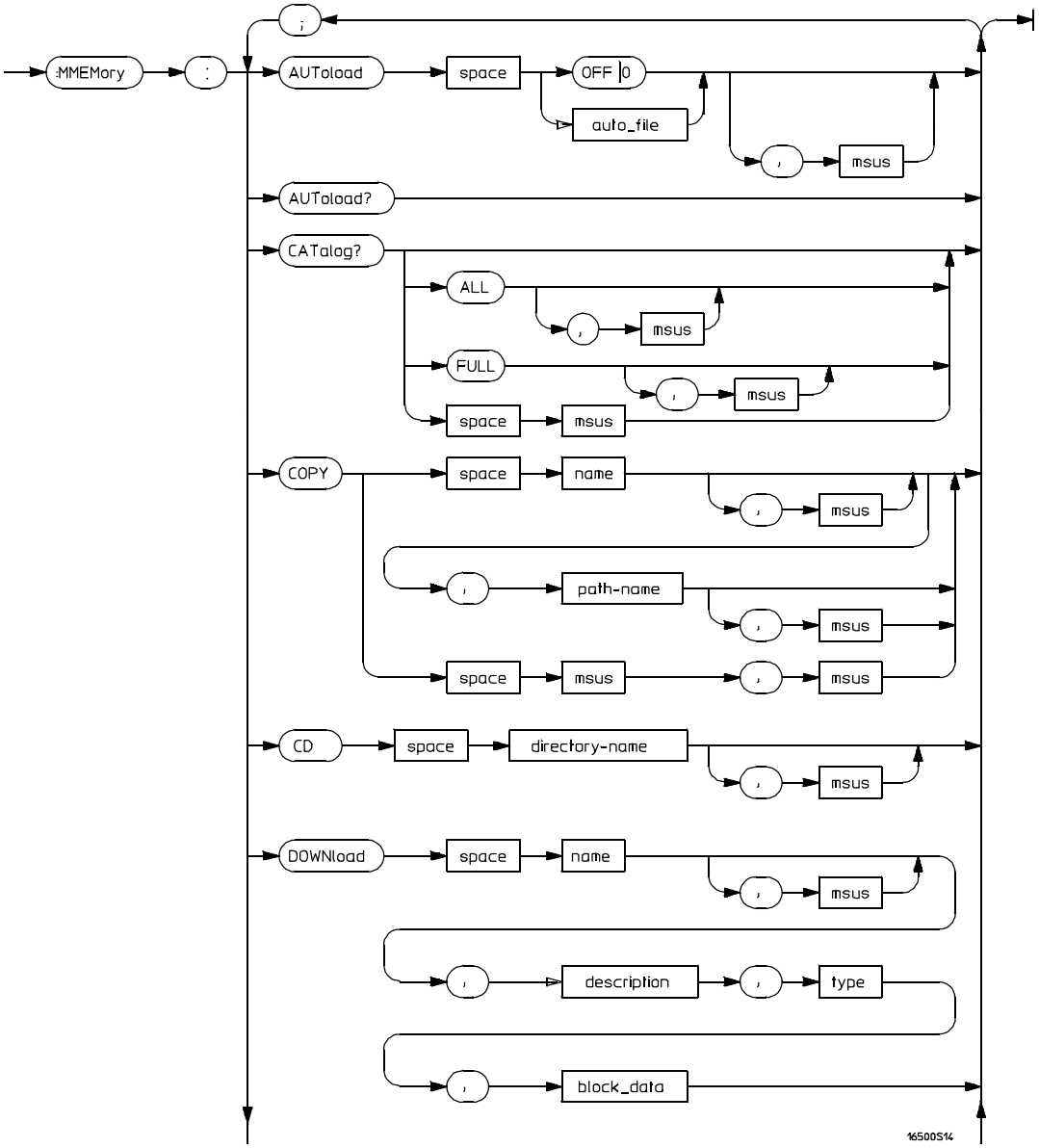
Refer to figure 12-1 and table 12-1 for the MMEMory Subsystem commands syntax diagram. The MMEMory subsystem commands are:

- AUToload
- CATalog
- CD (change directory)
- COPY
- DOWNload
- INITialize
- LOAD
- MKDir (make directory)
- MSI
- PACK
- PURGe
- PWD (present working directory)
- RENname
- STORe
- UPLoad
- VOLume

<msus> refers to the mass storage unit specifier. INTernal0 specifies the hard disk drive and INTernal1 specifies the flexible disk drive.

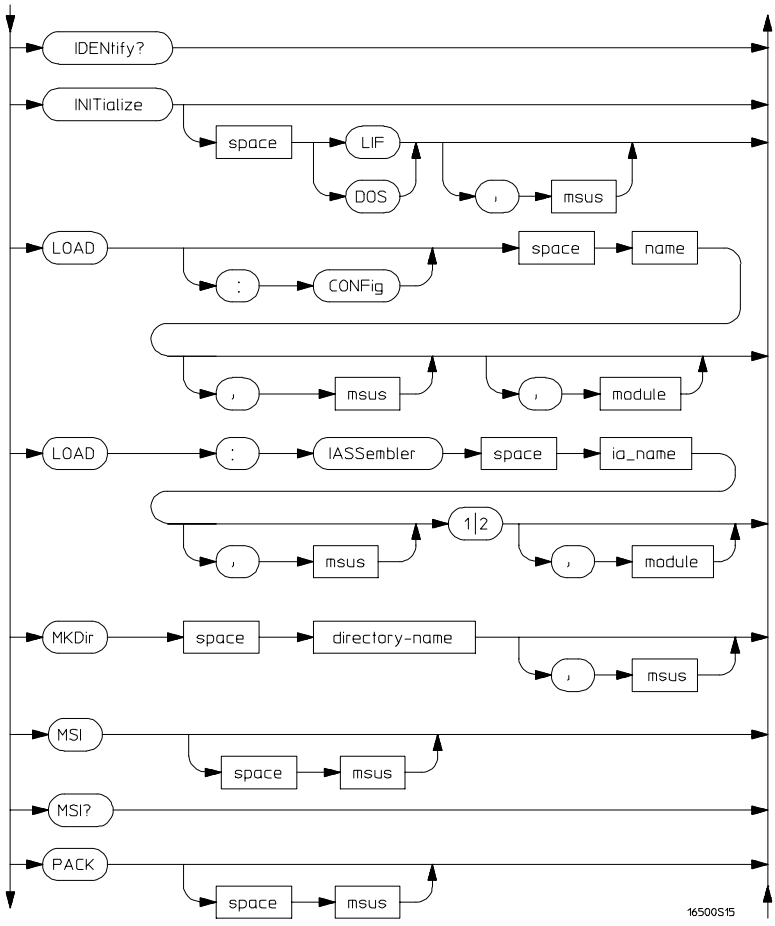
If you are not going to store information to the flexible configuration disk, or if the flexible disk you are using contains information you need, it is advisable to write protect your disk. This will protect the contents of the disk from accidental damage due to incorrect commands being mistakenly sent.

Figure 12-1



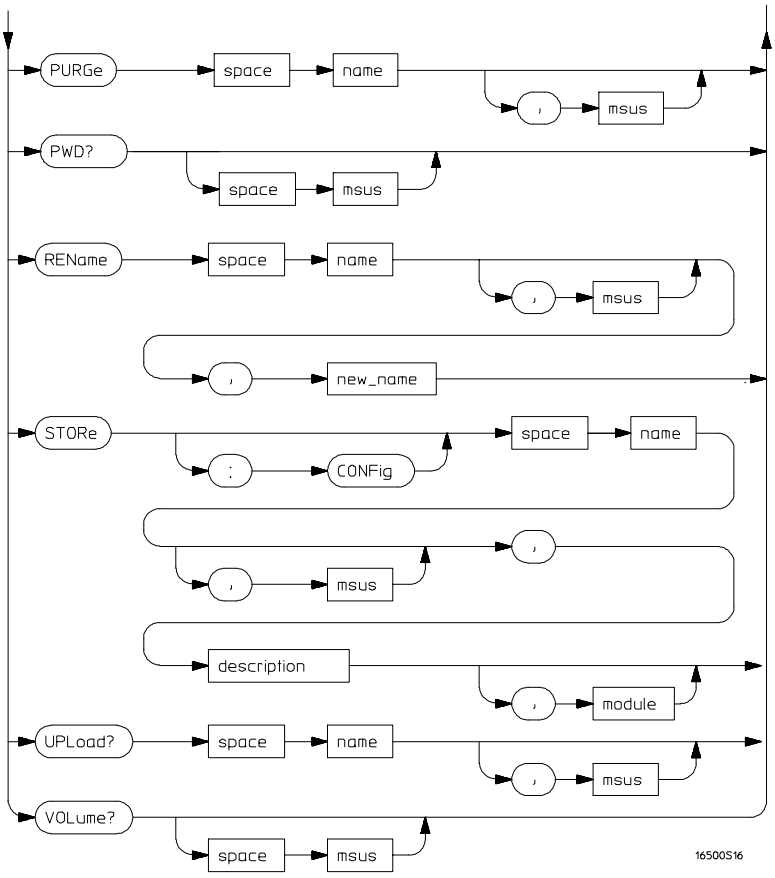
MMEMory Subsystem Commands Syntax Diagram

Figure 12-1 (Continued)



MMEMory Subsystem Commands Syntax Diagram (Continued)

Figure 12-1 (Continued)



MMEMory Subsystem Commands Syntax Diagram (Continued)

Table 12-1 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. **INTernal0** for the hard disk drive and **INTernal**1 for the flexible disk drive.
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"
path_name	A string of up to 64 characters for DOS disks ending in a file name. Separators can be the slash (/) or the backslash (\) character.
directory_name	A string of up to 64 characters for DOS disks ending in a directory name. Separators can be the slash (/) or the backslash (\) character. The string of two periods (..) represents the parent of the present working directory.
description	A string of up to 32 alphanumeric characters.
type	An integer, refer to table 12-2.
block_data	Data in IEEE 488.2 format.
module	An integer, –2 through 5 for the HP 16500C alone or –2 through 10 with the HP 16501A connected.
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"

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 drive, the autoload feature is enabled to the specified file. The configuration files specified must reside in the root directory of the current drive.

`<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. **INTernal0** for the hard disk drive and **INTernal1** for the flexible disk drive.

Example Disabling autoload:

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

Setting FILE2 on the hard drive as the autoload file:

```
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.

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

Example

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

CATalog

Query `:MMEMory:CATalog? [[ALL|FULL][,<msus>]]`

> The FULL option is available with version 1.01 or higher of the HP 16500C operating system only. Version 1.00 does not recognize this option.

The CATalog query returns the directory of the disk in one of three block data formats. When no options are used, the directory consists of a string of 51 characters for each file on the disk. Each entry is formatted as follows:
`"NNNNNNNNNN TTTTTTT FFFFFFFFFFFFFFFFFFFFFFFFFFFFFF"`
where N is the filename, T is the file type (see table 12-2), and F is the file description.

The optional parameter **`ALL`** returns the directory of the disk as a 70-character string for each file, formatted as follows:
`"NNNNNNNNNNNN TTTTTTT FFFFFFFFFFFFFFFFFFFFFFFFFFFFFF DDMMMYY HH:MM:SS"`
where D, M, Y, and HH:MM:SS are respectively the date, month, year, and time in 24-hour format.

The optional parameter `FULL` returns an 83-character string for each file in the directory. The string is formatted as follows:
`"NNNNNNNNNNNN TTTTTTT FFFFFFFFFFFFFFFFFFFFFFFFFFFFFF DDMMMYY HH:MM:SS LLLLLLLLLLL"`
where L is the file length in decimal.

`<msus>` Mass Storage Unit specifier. **`INTernal0`** for the hard disk drive and **`INTernal`**`1` for the flexible disk drive.

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

`<block_data>` ASCII block containing
<**`filename> <file_type> <file_description>`**
in one of the three formats described above.

Example

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

CD (Change Directory)

Command

```
:MMEMory:CD <directory_name> [,<msus>]
```

The CD command allows you to change the current working directory on the hard disk or a DOS flexible disk. The command allows you to send path names of up to 64 characters for DOS format. Separators can be either the slash (/) or backslash (\) character. Both the slash and backslash characters are equivalent and are used as directory separators. The string containing double periods (..) represents the parent of the directory.

`<directory_name>` String of up to 64 characters for DOS disks ending in the new directory name.

`<msus>` Mass Storage Unit specifier. **`INTernal0`** for the hard disk drive and **`INTernal1`** for the flexible disk drive.

Example

```
OUTPUT XXX;":MMEMory:CD 'CHILD_DIR'"
OUTPUT XXX;":MMEMory:CD '..'"
OUTPUT XXX;":MMEMory:CD '\SYSTEM\SOURCE_DIR\DIR', INTernal0"
```

The slash (/) character in DOS path names will be automatically translated to the backslash character (\) on the disk; therefore, any flexible DOS disk used in the HP 16500C will be compatible in DOS computers.

COPY

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

The COPY command copies one file to a new file. Wildcards are supported. The two **<name>** parameters are the filenames. The first pair of parameters specifies the source file. The second pair specifies the destination. An error is generated if the source file doesn't exist, or if the destination file already exists. The destination may be a directory, in which case the new file name is the same as the source file name.

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

NNNNNNNNNN
or
A string of up to 64 alphanumeric characters for DOS in one of the following forms:

NNNNNNNN.NNN when the file resides in the current directory or
\NAME_DIR\FILENAME when it does not reside in the current directory

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

NNNNNNNNNN
or
A string of up to 64 alphanumeric characters for DOS in one of the following forms:

NNNNNNNN.NNN when the file resides in the current directory or
\NAME_DIR to copy the file to another directory
or
\NAME_DIR\FILENAME to copy the file to another directory and change the name.

`<msus>` Mass Storage Unit specifier. **INTernal0** for the hard disk drive and **INTernal1** for the flexible disk drive.

Example

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

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

To copy the contents of "FILE1" on the hard disk to "FILE2" on the flexible disk:

```
OUTPUT XXX;":MMEMORY:COPY 'FILE1',INTERNAL0,'FILE2',INTERNAL1"
```

To copy the contents of a LIF flexible disk to the hard disk:

```
OUTPUT XXX;":MMEM:COPY '*',INT1,'\',INT0"
```

DOWNload

Command

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

The DOWNload command downloads data to a file on the mass storage device. The **<block_data>** parameter contains the data; the <name> parameter is the name of the file being created.

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

NNNNNNNNNN
or
A string of up to 64 alphanumeric characters for DOS in one of the following forms:

NNNNNNNN.NNN when the file resides in the current directory or
\NAME_DIR\FILENAME when it does not reside in the current directory

<msus> Mass Storage Unit specifier. **INTernal0** for the hard disk drive and **INTernal1** for the flexible disk drive.

<description> A string of up to 32 alphanumeric characters

<type> An integer (see table 12-2)

<block_data> Contents of file in block data format

Example

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

Table 12-2 File Types

File	File Type
Autoload File	–15615
Inverse Assembler	–15614
DOS file (from Print to Disk)	–5813
HP 16500B System Software	–15603
HP 16500B Option Software	–15602
HP 16500C System Software	–15593
HP 16500C Option Software	–15592
HP 16500A/B/C System Configuration	–16127
HP 16510A/B Configuration	–16096
HP 16511B Configuration	–16097
HP 16515A (Master) and 16516A (Expander) Configuration	–16126
HP 16517A (Master) and 16518A (Expander) Configuration	–16123
HP 16520A (Master) and 16521A (Expander) Configuration	–16106
HP 16522A Configuration	–16102
HP 16530A (Timebase) and 16531A (Acquisition) Configuration	–16116
HP 16532A Configuration	–16114
HP 16533A and 16534A Configuration	–16113
HP 16535A Configuration	–16112
HP 16540A/D (Master) and 16541A/D (Expander) Configuration	–16087
HP 16542A Configuration	–16085
HP 16550A Configuration	–16095
HP 16554A, 16555A/D, and 16556A/D Configuration	–16093

IDENtify

Command

```
:MMEMory:IDENtify? [<msus>]
```

> The IDENtify query is available with version 1.01 or higher of the HP 16500C operating system only. Version 1.00 does not recognize this query.

The IDENtify query returns the serial number of the disk in the specified drive. This number is unique for disks created with DOS version 4.0 or higher. For other disks, the number returned is usually 0.

Returned Format

```
VOLUME SERIAL #: <serial_num>
```

> The IDENtify query always returns the initial string "VOLUME SERIAL #: " and has no response header.

`<msus>` Mass Storage Unit specifier. **`INTernal0`** for the hard disk drive and **`INTernal1`** for the flexible disk drive.

`<serial_num>` An eight-digit hexadecimal number

INITialize

Command

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

The INITialize command formats the disk in DOS (Disk Operating System) on the hard drive or either DOS or LIF (Logical Information Format) on the flexible drive. If no format is specified, then the initialize command will format the disk in the DOS format. LIF format is not allowed on the hard drive.

`<msus>` Mass Storage Unit specifier. **`INTernal0`** for the hard disk drive and **`INTernal`**`1` for the flexible disk drive.

Example

```
OUTPUT XXX;":MMEMORY:INITIALIZE DOS"
OUTPUT XXX;":MMEMORY:INITIALIZE LIF,INTERNAL1"
OUTPUT XXX;":MMEM:INIT DOS,INT0"
```

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 modules, 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 –2 through 10. 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 the modules, 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 64 alphanumeric characters for DOS in one of the following forms:

NNNNNNNN.NNN when the file resides in the current directory or **\NAME_DIR\FILENAME** when it does not reside in the current directory

`<msus>` Mass Storage Unit specifier. **INTernal0** for the hard disk drive and **INTernal1** for the flexible disk drive.

`<module>` An integer, –2 through 5 for the HP 16500C alone. –2 through 10 with the HP 16501A connected.

Example

```
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. For example, a 1 specifies that the inverse assembler files will be loaded into MACHINE 1 of the specified module.

The optional **<module>** parameter is used to specify which slot the state analyzer is in. If this parameter is not specified, the state analyzer in the currently selected module will be loaded with the inverse assembler file.

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

NNNNNNNNNN
or
A string of up to 64 alphanumeric characters for DOS in one of the following forms:

NNNNNNNN.NNN when the file resides in the current directory or **\NAME_DIR\FILENAME** when it does not reside in the current directory

<msus> Mass Storage Unit specifier. **INTernal0** for the hard disk drive and **INTernal1** for the flexible disk drive.

<module> An integer, 1 through 5 for the HP 16500C alone. 1 through 10 with an HP 16501A connected. Although the parser will accept the values –1 and –2, they will generate a disk error.

Example

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

MKDir (Make Directory)

Command

```
:MMEMory:MKDir <directory_name> [,<msus>]
```

The MKDir command allows you to make a directory on the hard drive or a DOS disk in the flexible drive. Directories cannot be made on LIF disks. Make directory will make a directory under the present working directory on the current drive if the optional path is not specified. Separators can be either the slash (/) or backslash (\) character. Both the slash and backslash characters are equivalent and are used as directory separators. The string containing two periods (..) represents the parent of the present working directory.

`<directory_name>` String of up to 64 characters for DOS disks ending in the new directory name.

`<msus>` Mass Storage Unit specifier. **`INTernal0`** for the hard disk drive and **`INTernal1`** for the flexible disk drive.

Example

```
OUTPUT XXX;":MMEMORY:MKDIR 'NEW.DIR'"
OUTPUT XXX;":MMEM:MKD '\SYSTEM\NEW.DIR',INT0 "
```

The slash (/) character in DOS path names will be automatically translated to the backslash character (\) on the disk; therefore, any flexible DOS disk used in the HP 16500C will be compatible in DOS computers.

MSI (Mass Storage Is)

Command

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

The MSI command selects a default mass storage device. INTernal0 selects the hard disk drive and INTernal1 selects the flexible disk drive. Once the MSI is selected it remains the default drive until another MSI command is sent to the system.

`<msus>` Mass Storage Unit specifier. **`INTernal0`** for the hard disk drive and **`INTernal`**`1` for the flexible disk drive.

Example

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

Query

```
:MMEMory:MSI?
```

The MSI? query returns the current MSI setting, either INTernal0 or INTernal1.

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. **`INTernal0`** for the hard disk drive and **`INTernal`**`1` for the flexible disk drive.

Example

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

PURGe

Command

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

The PURGe command deletes files and directories from the disk in the specified drive. The PURge command only purges directories when the directory is empty. If the PURge command is sent with a directory name and the directory contains files, the message "Directory contains files" is displayed and the command is ignored. The **<name>** parameter specifies the file name 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 64 alphanumeric characters for DOS in one of the following forms:

NNNNNNNN.NNN when the file resides in the current directory or **\NAME_DIR\FILENAME** when it does not reside in the current directory

<msus> Mass Storage Unit specifier. **INTernal0** for the hard disk drive and **INTernal1** for the flexible disk drive.

Example

This instruction purges the file named "FILE1" from the currently specified drive:

```
OUTPUT XXX;":MMEMORY:PURGE 'FILE1'"
```

This purges all files in the directory named "TEMP" from the hard drive:

```
OUTPUT XXX;":MMEMORY:PURGE '\TEMP\*.*',INTERNAL0"
```

This instruction purges the directory named "TEMP" from the hard drive:

```
OUTPUT XXX;":MMEMORY:PURGE '\TEMP',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.

PWD (Present Working Directory)

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

The PWD query returns the present working directory for the specified drive. If the **<msus>** option is not sent, the present working directory will be returned for the current drive.

Returned Format `[:MMEMory:PWD] <directory>,<msus><NL>`

`<directory>` String of up to 64 characters with the backslash (\) as separator for DOS and LIF disks.

`<msus>` Mass Storage Unit specifier. **`INTernal0`** for the hard disk drive and **`INTernal`**`1` for the flexible disk drive.

Example

```
OUTPUT XXX;":MMEMORY:PWD?"
OUTPUT XXX;":MMEMORY:PWD? INTERNAL1"
```

REName

Command

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

The REName command renames a file on the drive. The **<name>** parameter specifies the filename to be changed and the **<new_name>** parameter specifies the new filename. You cannot use REName to move a file from one drive to the other.

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 64 alphanumeric characters for DOS in one of the following forms:

NNNNNNNN.NNN when the file resides in the current directory or **\NAME_DIR\FILENAME** when it does not reside in the current directory

<msus> Mass Storage Unit specifier. **INTernal0** for the hard disk drive and **INTernal1** for the flexible disk drive.

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

NNNNNNNNNN
or
A string of up to 64 alphanumeric characters for DOS in one of the following forms:

NNNNNNNN.NNN when the file resides in the current directory or **\NAME_DIR\FILENAME** when it does not reside in the current directory

Example

```
OUTPUT XXX;":MMEMORY:RENAME 'AUTOLOAD','OLD_AUTO'"
```

STORe [:CONFig]

Command

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

The STORe command stores 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 modules. If the optional **<module>** parameter is not specified, the configurations for the system, modules, and any installed software options are stored.

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

NNNNNNNNNN
or
A string of up to 64 alphanumeric characters for DOS in one of the following forms:

NNNNNNNN.NNN when the file resides in the current directory or
\NAME_DIR\FILENAME when it does not reside in the current directory

`<msus>` Mass Storage Unit specifier. **INTernal0** for the hard disk drive and **INTernal1** for the flexible disk drive.

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

`<module>` An integer, –2 through 5 for the HP 16500C alone. –2 through 10 with an HP 16501A connected.

Example

```
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 through E) for an HP 16500C alone or an _(A through J) with an HP 16501A connected.

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 configuration files.

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

NNNNNNNNNN

or

A string of up to 64 alphanumeric characters for DOS in one of the following forms:

NNNNNNNN.NNN when the file resides in the current directory or **\NAME_DIR\FILENAME** when it does not reside in the current directory

`<msus>` Mass Storage Unit specifier. **INTernal0** for the hard disk drive and **INTernal**1 for the flexible disk drive.

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:HEADER 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>]`

The VOLume 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. **`INTernal0`** for the hard disk drive and **`INTernal1`** for the flexible disk drive.

> The VOLume query does not return a response header.

Returned Format `{DOS|LIF|???}<NL>`

Example

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

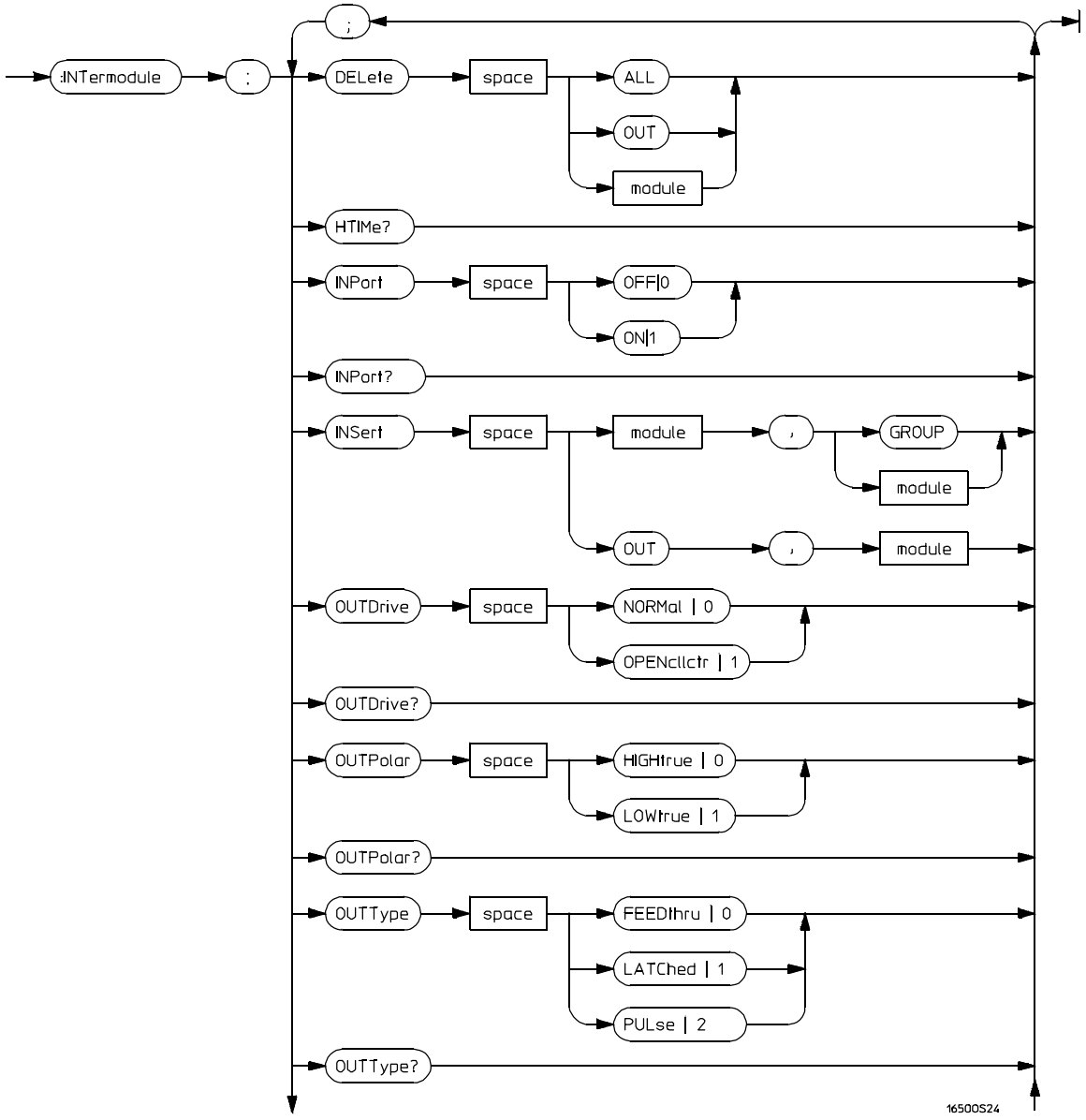
INTermodule Subsystem

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 13-1 and table 13-1 for the INTermodule Subsystem commands syntax diagram. The INTermodule commands are:

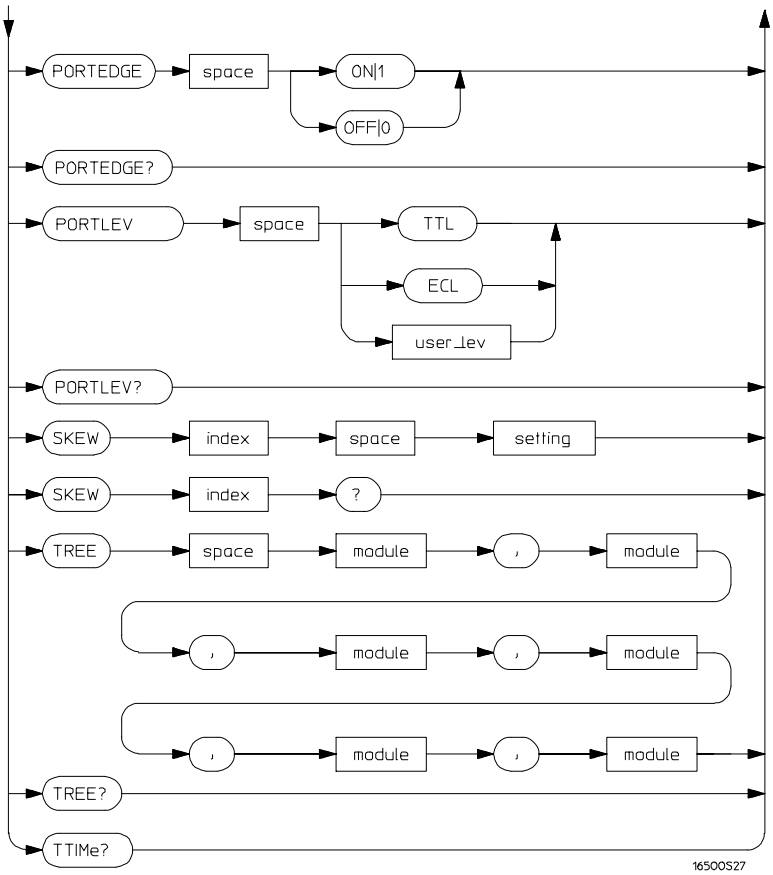
- DELete
- HTIMe
- INPort
- INSert
- OUTDrive
- OUTPolar
- OUTType
- PORTEDGE
- PORTLEV
- SKEW
- TREE
- TTIMe

Figure 13-1



Intermodule Subsystem Commands Syntax Diagram

Figure 13-1 (continued)



Intermodule Subsystem Commands Syntax Diagram (continued)

Table 13-1 **INTermodule Parameter Values**

Parameter	Value
module	An integer, 1 through 5 for HP 16500C alone. 1 through 10 with the HP 16501A connected.
user_lev	A real number from –4.0 to +5.0 volts in 0.02 volt increments
index	An integer, 1 through 5 for HP 16500C alone. 1 through 10 with the HP 16501A connected.
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

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

The DELete command is used to remove a module, PORT OUT, or an entire intermodule tree from a Group Run. The **<module>** parameter sent with the delete command refers to the slot location of the module.

`<module>` An integer, 1 through 5 for HP 16500C alone. 1 through 10 with the HP 16501A connected.

Example

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

HTIMe

Query `:INTermodule: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. These values are for reference only because the values returned by TTIMe include the internal hardware skew represented by HTIMe.

Returned Format `[:INTermodule:HTIMe] <value_A>,<value_B>,<value_C>,<value_D>,<value_E>[,<value_F>,<value_G>,<value_H>,<value_I>,<value_J>]<NL>`

`<value_X>` Skew for module in slot X (real number)

Example

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

INPort

Command

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

The INPort command causes intermodule acquisitions to be armed from the Port In, the same as Group Run Armed from PORT IN in the Intermodule menu. A value of 0 removes Port In from the Group Run.

In version 1.02 and later of the operating system software, you can set Group Run with OR TRIGGER by setting INPort to 2.

Example

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

Query

```
:INTermodule:INPort?
```

The INPort query returns the current Group Run setting. A value of 2 means that the Group Run is set to Group Run with OR TRIGGER.

Returned Format

```
[:INTermodule:INPort] {2|1|0}<NL>
```

Example

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

INSert

Command

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

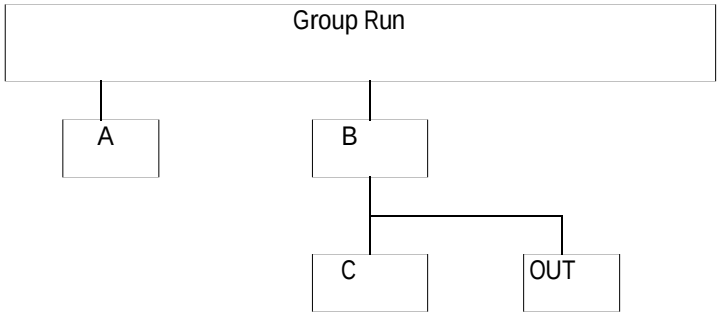
The INSert command adds a module or PORT OUT to the Intermodule configuration. The first parameter selects the module or PORT OUT to be added to the intermodule configuration, and the second parameter tells the instrument where the module or PORT OUT will be located. 1 through 5 corresponds to the slot location of the modules A through E for the HP 16500C alone and 1 through 10 corresponds to slot location of modules A through J when an HP 16501A is connected.

`<module>` An integer, 1 through 5 for HP 16500C alone. 1 through 10 with the HP 16501A connected.

Example

```
OUTPUT XXX;":INTERMODULE:INSERT 1,GROUP"
OUTPUT XXX;":INTERMODULE:INSERT 2,GROUP"
OUTPUT XXX;":INTERMODULE:INSERT 3,2;INSERT OUT,2"
```

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



OUTDrive

Command

```
:INTermodule:OUTDrive {{0|NORMal}|{1|OPENcllctr}}
```

The OUTDrive command sets the Port Out BNC to put out either a normal (TTL-type) or open-collector signal. This corresponds to the Output field in the PORT IN/OUT Setup menu under the Intermodule menu.

See Also

The *HP 16500C User's Reference* for more information about open collector signals.

Example

```
OUTPUT XXX;":INTERMODULE:OUTDRIVE NORMAL"
```

Query

```
:INTermodule:OUTDrive?
```

The OUTDrive query returns the current Port Out output setting.

Returned Format

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

Example

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

OUTPolar

Command

```
:INTermodule:OUTPolar {{0|HIGHtrue}|{1|LOWtrue}}
```

The OUTPolar command sets the Port Out BNC polarity. This command has the same effect as setting the Polarity field in the PORT IN/OUT Setup menu under the Intermodule menu.

Example

```
OUTPUT XXX;":INTERMODULE:OUTP HIGH"
```

Query

```
:INTermodule:OUTPolar?
```

The OUTPolar query returns the current Port Out polarity setting.

Returned Format

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

Example

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

OUTType

Command

```
:INTermodule:OUTType {{0|FEEDthru} | {1|LATChed}
| {2|PULse}}
```

The OUTType command sets the Port Out BNC signal type. This command has the same effect as setting the Type field in the PORT IN/OUT Setup menu under the Intermodule menu.

See Also

The *HP 16500C User's Reference* for more information about Port Out signal types.

Example

```
OUTPUT XXX;":INTERMODULE:OUTTYPE LATCHED"
```

Query

```
:INTermodule:OUTType?
```

The OUTType query returns the current Port Out signal type.

Returned Format

```
[:INTermodule:OUTType] {0|1|2}<NL>
```

Example

```
OUTPUT XXX;":INT:OUTT?"
```

PORTEDGE

Command

```
:INTermodule:PORTEDGE <edge_spec>
```

This command does not obey the truncation rules.

The PORTEDGE command sets the Port In BNC to respond to either a rising edge of falling edge for a trigger from an external source. The threshold level of the input signal is set by the PORTLEV command.

`<edge_spec>` A 1 or ON for rising edge or a 0 or OFF for falling edge.

Example

```
OUTPUT XXX;":INTERMODULE:PORTEDGE 1"
```

Query

```
:INTermodule:PORTEDGE?
```

The PORTEDGE query returns the current edge setting.

Returned Format

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

Example

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

PORTLEV

Command

```
:INTermodule:PORTLEV {TTL|ECL|<user_lev>}
```

This command does not obey the truncation rules.

The PORTLEV (port level) command sets the threshold level at which the input BNC responds and produces an intermodule trigger. The preset levels are TTL and ECL. The user-defined level is –4.0 volts to +5.0 volts. If a value outside this range is specified, the value is set to the extreme in the direction exceeded and no error message is generated.

`<user_lev>` A real number from –4.0 to + 5.0 volts in 0.01 volt increments.

Example

This statement sets the BNC threshold to ECL

```
OUTPUT XXX;":INTERMODULE:PORTLEV ECL"
```

This statement sets the BNC threshold to –2.3 volts

```
OUTPUT XXX;":INTERMODULE:PORTLEV -2.3"
```

Query

```
:INTermodule:PORTLEV?
```

The PORTlev query returns the current BNC threshold setting.

Returned Format

```
[INTermodule:PORTLEV] {TTL|ECL|<user_lev><NL>
```

Example

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

SKEW<N>

Command

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

The SKEW command sets the skew value for a module. The <N> index value is the module number (1 through 5 corresponds to the slot location of the modules A through E for the HP 16500C alone and 1 through 10 to slot location of modules A through J when an HP 16501A is connected). The <setting> parameter is the skew setting (– 1.0 to 1.0) in seconds.

`<N>` An integer, 1 through 5 for HP 16500C alone. 1 through 10 with the HP 16501A connected.

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

Example

```
OUTPUT XXX;":INTERMODULE:SKEW2 3.0E-9"
```

Query

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

The query returns the user defined skew setting.

Returned Format

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

Example

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

TREE

Command

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

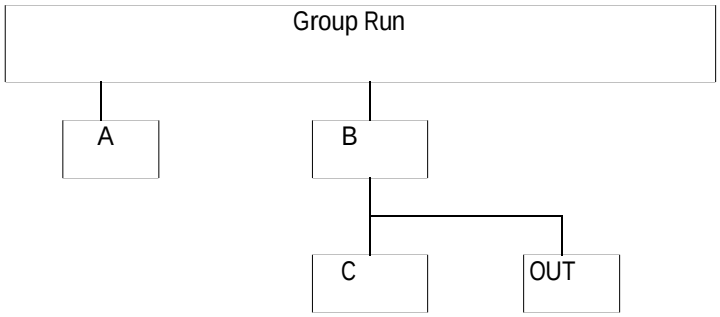
The TREE command allows an intermodule setup to be specified in one command. The first five (or ten, with an HP 16501A connected) parameters are the intermodule arm values for modules A through E for an HP 16500C alone or modules A through J with an HP 16501A connected. The last parameter corresponds 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. 1 through 10 correspond to slots A through J.

`<module>`
`<port_out>` An integer, –1 through 5 for an HP 16500C alone. –1 through 10 with an HP 16501A connected.

Example

```
OUTPUT XXX;":INTERMODULE:TREE 0,0,2,-1,-1,2"
```

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



Query

```
:INTermodule: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. 1 through 10 correspond to the slots A through J.

Returned Format

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

Example

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

TTIMe

Query `:INTermodule:TTIMe?`

The TTIMe query returns five values for the HP 16500C alone or ten with an HP 16501A connected representing the absolute intermodule trigger time for all of the modules in the Intermodule configuration. 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 has not yet been run;
- No module is installed in the corresponding slot;
- 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] <module>,<module>,<module>,<module>,<module>[,<module>,<module>,<module>,<module>,<module>]<NL>`

`<module>` Trigger time for module (real number)

Example

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

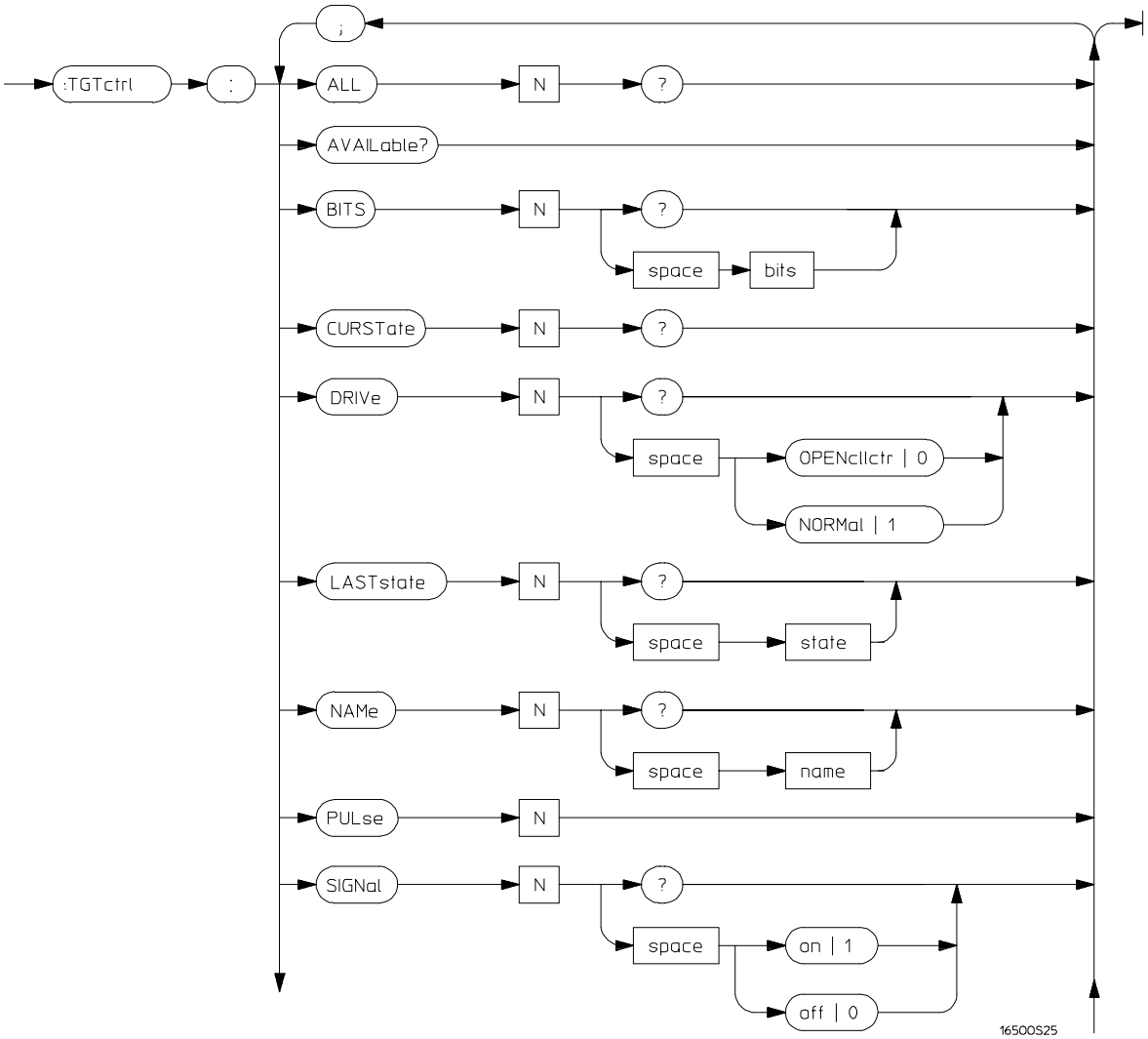
TGTctrl Subsystem

Introduction

The TGTctrl subsystem commands specify the signals put out by the Target Control Port. Refer to figure 14-1 and table 14-1 for the TGTctrl Subsystem commands syntax diagram. The TGTctrl commands are:

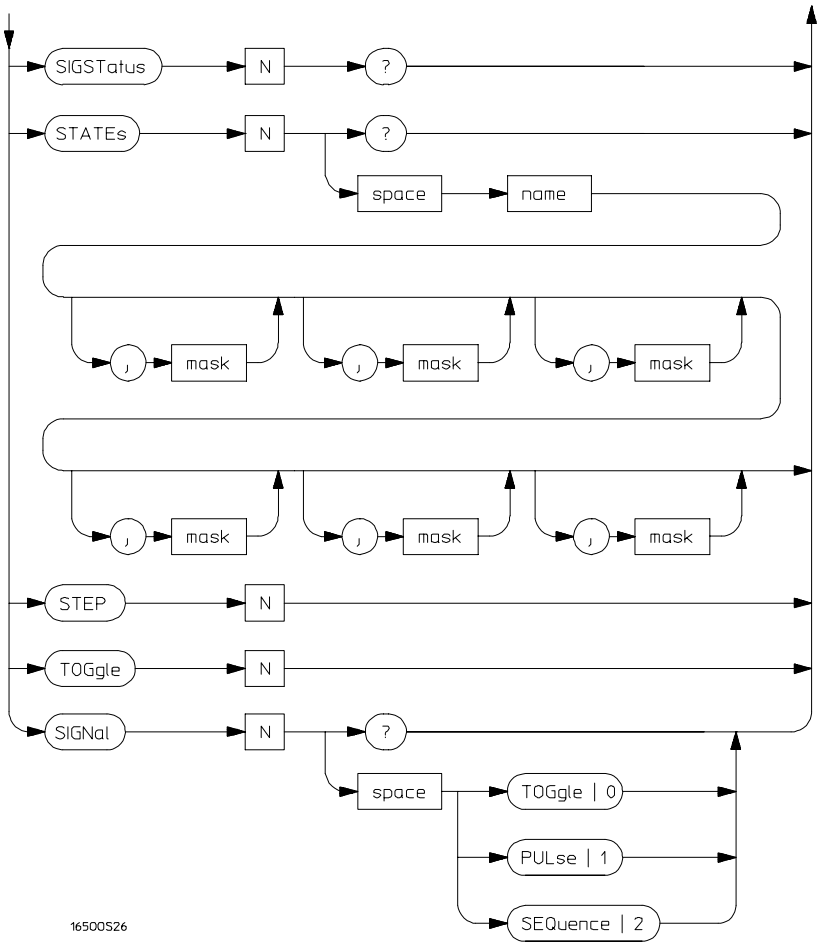
- ALL
- AVAILable
- BITS
- CURSTate
- DRIVe
- LASTstate
- NAME
- PULse
- SIGNal
- SIGSTatus
- STATEs
- STEP
- TOGgle
- TYPE

Figure 14-1



Targetcontrol Subsystem Commands Syntax Diagram

Figure 14-1 (continued)



Targetcontrol Subsystem Commands Syntax Diagram

Table 14-1 **TGTctrl Parameter Values**

Parameter	Value
N	An integer, 0 through 7, indicating signal
bits	An integer, 0 though 255
state	An integer, 0 through 7
name	A string of up to 14 characters

:TGTctrl

Selector `:TGTctrl`

This command does not obey the truncation rule.

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

Example

```
OUTPUT XXX;":TGTCTRL:ALL4?"
```

ALL

Query `:TGTctrl:ALL<N>?`

The ALL query returns all parameters of the signal specified by <N>. These values may be individually queried using other commands in the TGTctrl subsystem.

`<N>` An integer, 0 through 7.

Returned Format `[:TGTctrl:ALL<N>] <on>,<name>,<type>,<bits>,<drive>,<last>, <current>,<state0>,<state1>,<state2>,<state3>,<state4>, <state5>,<state6>,<state7><nl>`

`<on>` `{0|1}` 0 indicates signal is not active, 1 indicates signal is active.

`<name>` A string of up to 14 characters

`<type>` `{0|1|2}` indicating type of output signal. 0 indicates toggle, 1 indicates pulse, and 2 indicates a sequence.

`<bits>` An integer, 0 through 255, whose binary form indicates the signal's bit mask. A value of 1 means the bit is turned on.

`<drive>` `{0|1}` indicating output type. A value of 0 indicates normal (TTL), and 1 indicates open collector.

`<last>` An integer, 1 through 7, indicating last state.

`<current>` An integer, 0 through 7, indicating current state.

`<state0> – <state7>` An integer, 0 through 255, whose binary form indicates the value put out when the signal is in that state. Values occurring after the last state are not significant.

Example

```
OUTPUT XXX;"TGT:ALL0?"
```

See Also SIGNal, NAME, TYPE, BITS, DRIVe, LASTstate, CURSTate, and STATEs commands in this chapter.

AVAILable

Query `:TGTctrl:AVAILable?`

This command does not obey the truncation rule.

The AVAILable query returns an integer whose binary form indicates unassigned bits. If a signal is turned off, any bits assigned to it are available. A 0 indicates that the bit is NOT available.

Returned Format `[:TGTctrl:AVAILable] <bits><NL>`

`<bits>` An integer, 0 through 255.

Example

For example, if your HP BASIC program contains a line

```
OUTPUT XXX;":TGTCTRL:AVAILABLE?"
```

and the value returned is 248, you have 1111 1000 in binary. The binary value lets you know that bits 0 through 2 are in use, but bits 3 through 7 are available.

BITS

Command

```
:TGTctrl:BITS<N> <mask>
```

The BITS command assigns bits to a signal. A 1 in the mask's bit position assigns the bit to the signal. The mask overwrites any previous assignments.

`<N>` An integer, 0 through 7, specifying signal.

`<mask>` An integer, 0 through 255.

Example

To assign the last four bits to signal 1:

```
OUTPUT XXX;":TGT:BITS1 #H0F"
```

Query

```
:TGTctrl:BITS<N>?
```

The query returns an integer. The binary form of the integer specifys which bits are assigned to the signal.

Returned Format

```
[:TGTctrl:BITS<N>] <mask><NL>
```

Example

```
OUTPUT XXX;":TGT:BITS1?"
```

CURSTate

Query `:TGTctrl:CURSTate<N>?`

This command does not obey the truncation rule.

The CURSTate query returns the current state of the specified signal. For toggle and pulse signals, this will be either 0 or 1. For sequence signals, it will be between 0 and the last state.

Returned Format `[:TGTctrl:CURSTate<N>] <state><NL>`

`<N>` An integer, 0 through 7, specifying signal.

`<state>` An integer, 1 through 7, indicating current state.

Example

```
OUTPUT XXX;":TGTCTRL:CURSTATE0?"
```

DRIVe

Command `:TGTctrl:DRIVe<N> {{NORMal|0}|{OPENcllctr|1}}`

The DRIVe command sets a signal's output type. The output type controls the way in which the line is driven.

`<N>` An integer, 0 through 7, specifying signal.

Example

```
OUTPUT XXX;":TGTCTRL:DRIVE1 NORMAL"
```

Query `:TGTctrl:DRIVe<N>?`

Returned Format `[:TGTctrl:DRIVe<N>] <type><NL>`

`<type>` `{0|1}` 0 indicates normal (TTL); 1 indicates open collector.

LASTstate

Command `:TGTctrl:LASTstate<N> <state>`

The LASTstate command sets a signal's last state. LASTstate has no effect unless the signal type is sequence.

`<N>` An integer, 0 through 7, specifying signal.

`<state>` An integer, 1 through 7.

Example

```
OUTPUT XXX;":TGTCTRL:LAST2 4"
```

Query `:TGTctrl:LASTstate<N>?`

The query returns the current last state. For toggle and pulse signals, the last state is always 1.

Returned Format `[:TGTctrl:LASTstate<N>] <state><NL>`

Example

```
OUTPUT XXX;":TGTCTRL:LAST1?"
```

NAMe

Command `:TGTctrl:NAMe<N> <name>`

The LASTstate command sets a signal name.

`<N>` An integer, 0 through 7, specifying signal.

`<name>` A string of up to 14 characters.

Example

```
OUTPUT XXX;":TGTCTRL:NAME0 'Reset'"
```

Query `:TGTctrl:NAMe<N>?`

The NAMe query returns the name of the specified signal. The default names are SIGNAL 0 through SIGNAL 7.

Returned Format `[:TGTctrl:NAMe<N>] <name><NL>`

Example

```
OUTPUT XXX;":TGTCTRL:NAME0?"
```

PULse

Command `:TGTctrl:PULse<N>`

This command does not obey the truncation rule.

The PULse command pulses the specified signal. If the signal type is toggle or sequence, it sets the signal to the next state. This command works the same as STEP and TOGgle.

`<N>` An integer, 0 through 7, specifying signal.

Example

```
OUTPUT XXX;":TGTCTRL:PULSE7"
```

SIGNal

Command `:TGTctrl:SIGNal<N> {{OFF|0}|{ON|1}}`

The SIGNal command activates or deactivates the specified signal, like manually selecting Turn Signal Off or Turn Signal On in the Target Control Port Settings menu.

`<N>` An integer, 0 through 7, specifying signal.

Example

```
OUTPUT XXX;":TGT:SIGN7 ON"
```

Query `:TGTctrl:SIGNal<N>?`

The SIGNal query returns the current status of the specified signal.

Returned Format `[:TGTctrl:SIGNal<N>] <status><NL>`

`<status>` `{0|1}` 0 indicates off and 1 indicates on.

SIGSTatus

Query `:TGTctrl:SIGSTatus<N>?`

This command does not obey the truncation rule.

The SIGSTatus query returns two values. The first is the current state on the target control lines assigned to the signal. The second is an activity indicator on the signal. A signal that is off will always return a first value indicating all assigned lines are 1.

`<N>` An integer, 0 through 7, specifying signal.

Returned Format `[:TGTctrl:SIGSTatus<N>] <cur_st>,<activity><NL>`

`<cur_st>` An integer, from 0 to 255. The binary format represents the current state of the assigned lines. The return value is packed; that is, unassigned lines do not appear in the binary form.

`<activity>` An integer, from 0 to 255, representing recent activity. A 1 indicates that the bit shows activity. A 0 indicates that the bit has been stable since last screen update.

Example

```
OUTPUT XXX;":TGTCTRL:SIGSTATUS 3?"
```

STATEs

Command

```
:TGTctrl:STATEs<N> <value_0>[,<value_1>,
<value_2>,<value_3>,<value_4>,<value_5>,<value_6>,
<value_7>]
```

This command does not obey the truncation rule.

The STATEs command sets the state values of the specified signal. The value is interpreted as a packed bitmask. You can specify up to 8 values on the command line.

`<N>` An integer, 0 through 7, specifying signal.

`<value_N>` An integer, 0 through 255, specifying that state's value in a packed bitmask.

Example

If bits 0, 2, and 7 are assigned to signal 0, the following command would set each bit high in turn:

```
OUTPUT XXX;":TGT:STATE0 1,2,4"
```

Query

```
:TGTctrl:STATEs<N>?
```

The STATEs query returns values for all 8 states. For toggle and pulse signals, only the first two values are significant.

Returned Format

```
[:TGTctrl:STATEs<N>] <value_0>,<value_1>,<value_2>,
<value_3>,<value_4>,<value_5>,<value_6>,<value_7><NL>
```

Example

```
OUTPUT XXX;":TGTCTRL:STATES0?"
```

STEP

Command `:TGTctrl:STEP<N>`

The STEP command sets the specified signal to the next state. If the signal type is pulse, it briefly pulses the signal. STEP can be used with any type of signal. This command has the same effect as PULse and TOGgle.

`<N>` An integer, 0 through 7, specifying signal.

Example

```
OUTPUT XXX;":TGTCTRL:STEP3"
```

TOGgle

Command `:TGTctrl:TOGgle<N>`

This command does not obey the truncation rule.

The TOGgle command toggles the specified signal to the next state. If the signal type is pulse, it briefly pulses the signal. This command has the same effect as PULse and STEP.

`<N>` An integer, 0 through 7, specifying signal.

Example

```
OUTPUT XXX;":TGTCTRL:TOGGLE3"
```

TYPe

Command

```
:TGTctrl:TYPe<N> {{TOGgle|0} | {PULse|1} |
{SEQuence|2}}
```

The TYPe command sets the signal type for the specified signal. It does not turn on the signal.

`<N>` An integer, 0 through 7, specifying signal.

Example

```
OUTPUT XXX;":TGT:TYP2 SEQ"
```

Query

```
:TGTctrl:TYPe<N>?
```

The TYPe query returns the current type of the signal. The default type is toggle.

Returned Format

```
[:TGTctrl:TYPe<N>] <type><NL>
```

`<type>` `{0|1|2}` where 0 indicates toggle, 1 indicates pulse, and 2 indicates sequence.

Part 3

15 Programming Examples 15-1

Programming Examples

15

Programming Examples

Introduction

This chapter contains short, usable, and tested program examples that cover the most frequently requested examples. The examples are written in HP BASIC 6.2.

- Transferring the mainframe configuration between the mainframe and the controller
- Checking for intermodule measurement completion
- Sending queries to the mainframe
- Getting ASCII data with PRINt? All query
- Reading a disk catalog
- Printing to the disk using PRINT? ALL

Transferring the Mainframe Configuration

This program uses the **SYSTem:SETup?** query to transfer the configuration of the mainframe to your controller. This program also uses the **SYSTem:SETup** command to transfer a mainframe configuration from the controller back to the mainframe. The configuration data will set up the mainframe according to the data. It is useful for getting configurations for setting up the mainframe by the controller. This command and query differs from the **SYSTem:DATA?** command and query because it only transfers the configuration and not the acquired data. Because the mainframe, by itself, does not acquire data the **SYSTem:DATA?** command and query is only useful for modules.

```
10    ! ****************** SETUP COMMAND AND QUERY EXAMPLE *******************
20    !                 for the HP 16500C/16501A Logic Analysis System
30    !
40    ! ********************* CREATE TRANSFER BUFFER ************************
50    ! Create a buffer large enough for the block data.
55    !
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 0"
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$(Byte);
440   Byte=Byte-48
450   IF Byte=1 THEN ENTER @Comm USING "#,D";Numbytes
460   IF Byte=2 THEN ENTER @Comm USING "#,DD";Numbytes
470   IF Byte=3 THEN ENTER @Comm USING "#,DDD";Numbytes
480   IF Byte=4 THEN ENTER @Comm USING "#,DDDD";Numbytes
490   IF Byte=5 THEN ENTER @Comm USING "#,DDDDD";Numbytes
500   IF Byte=6 THEN ENTER @Comm USING "#,DDDDDD";Numbytes
510   IF Byte=7 THEN ENTER @Comm USING "#,DDDDDDD";Numbytes
520   IF Byte=8 THEN ENTER @Comm USING "#,DDDDDDDD";Numbytes
530   PRINT Numbytes
540   !
550   ! ******************** TRANSER THE SETUP ******************************
560   ! Transfer the setup from the mainframe to the buffer.
570   !
580   TRANSFER @Comm TO @Buff;COUNT Numbytes,WAIT
600   ! Clear the terminator from the message queue
610   ENTER @Comm USING "-K";Length$
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 mainframe 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 16500C ******************
1020  ! Transfer the setup from the buffer to the HP 16500C mainframe.
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
```

Checking for Intermodule Measurement Completion

This program can be appended to or inserted into another program when you need to know when an intermodule 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.

```
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;":MESE0 1"
510   OUTPUT 707;":MESR0?"
520   ENTER 707;Status
530   !
533   ! Start a measurement
536   OUTPUT 707;":START"
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 Analysis System

This program example contains the steps required to send a query to the logic analysis system. Sending the query alone only puts the requested information in an output buffer of the logic analysis system. 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 analysis system, the query is properly terminated in the logic analyzer. If you send the query but fail to send an **ENTER** statement, the logic analysis system 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 16500C/16501A Logic analysis system
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 mainframe.
220   ! Always a 0 for the HP 16500C/16501A mainframe.
230   OUTPUT 707;":SELECT 0"
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 listing display, like the disk catalog or state listing, using the **PRINt? ALL** query. There are two things you must keep in mind:

- You must select the mainframe, which is always **SELECT 0** for the HP 16500C mainframe.
- You must select the proper menu. The only menus that allow you to use the **PRINt? ALL** query are the disk menu and listing menus.

```
10    !                  ****** ASCII DATA *******
20    !
30    !
40    ! This program gets the hard disk directory from the HP 16500C mainframe
50    !  in ASCII form by using the PRINT? ALL query.
60    !
70    !*************************************************************
80    !
90    DIM Block$[32000]
100   OUTPUT 707;"EOI ON"
110   OUTPUT 707;":SYSTEM:HEAD OFF"
120   OUTPUT 707;":SELECT 0"  ! Always a 0 for the HP 16500C mainframe
130   !
140   !
150   OUTPUT 707;":MENU 0,1"  ! Selects the hard disk menu.  Print? All
160                           ! will only work in disk menu and listings.
170   !
180   OUTPUT 707;":SYSTEM:PRINT? ALL"
190   ENTER 707 USING "-K";Block$
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 currently selected 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? ALL query
30    !
40    DIM File$[100]
50    DIM Specifier$[2]
60    OUTPUT 707;":EOI ON"
70    OUTPUT 707;":SYSTEM:HEADER OFF"
80    OUTPUT 707;":MMEMORY:MSI INTERNAL0" ! select the hard drive
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 70
170     ENTER 707 USING "#,70A";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 currently selected 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    OUTPUT 707;":MMEMORY:MSI INTERNAL0"  ! select the hard drive
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 flexible disk in the DOS format, 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 on a floppy disk.
50    ! It will print to either a LIF or DOS file using the PRINT ALL command.
60    !
70    !*************************************************************
80    ! This program assumes a logic analyzer module
85    ! is installed in slot 1.
90    OUTPUT 707;":SELECT 1"  ! Selects the module in slot 1. This program
100                           ! assumes a logic analyzer module is installed
110                           ! in slot 1.
115   !
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', INTERNAL1"
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"
235   OUTPUT 707;":MMEMORY:MSI INTERNAL1"
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

*CLS command, 9–5
*ESE command, 9–6
*ESR command, 9–7
*IDN command, 9–9
*IST command, 9–9
*OPC command, 9–11
*OPT command, 9–12
*PRE command, 9–13
*RST command, 9–14
*SRE command, 9–15
*STB command, 9–16
*TRG command, 9–17
*TST command, 9–18
*WAI command, 9–19
..., 5–5
32767, 5–4
9.9E+37, 5–4
::=, 5–5
, 5–5
[ ], 5–5
{ }, 5–5
|, 5–5

A
Addressed talk/listen mode, 2–3
ALL, 14–6
Angular brackets, 5–5
Arguments, 1–8
AUToload command, 12–7
AVAILable, 14–7

B
Bases, 1–13, 1–20
BASIC, 1–3
Baud rate, 3–8
BEEPer command, 10–6
Binary numbers, 1–13
Bit definitions, 7–4 to 7–5
BITS, 14–8
Block data, 1–7, 1–21
Block length specifier, 11–5, 11–12
Braces, 5–5
Bus addressing, 2–4

C
C programs
  See Examples
Cable
  RS-232-C, 3–3
CAPability, 10–7
Card identification numbers, 10–8
CARDcage command, 10–8
Case sensitivity, 6–9
CATalog, 12–8
CD, 12–9
CESE, 10–10
CESR, 10–11
Clear To Send (CTS), 3–5
Clock, 10–20
CME, 7–5
Comma, 1–13
Command, 1–7, 1–17
  *CLS, 9–5
  *ESE, 9–6
  *OPC, 9–11
  *PRE, 9–13
  *RST, 9–14
  *SRE, 9–15
  *TRG, 9–17
  *WAI, 9–19
  AUToload, 12–7
  BEEPer, 10–6
  BITS, 14–8
  CD (change directory, 12–9
  CESE, 10–10
  Combining, 1–10
  COPY, 12–10
  DATA, 11–5
  DELete, 13–6
  DOWNload, 12–11
  DRIVe, 14–9
  DSP, 11–6
  EOI, 10–13
  Errors, 8–3
  Execution, 6–3
  HEADer, 1–17, 11–8
  INITialize, 12–14
  INPort, 13–8
  INSert, 13–9
  LASTstate, 14–10
  LOAD:CONFig, 12–15
  LOAD:IASSembler, 12–16
  LOCKout, 3–10, 10–14
  LONGform, 1–17, 11–9
  MENU, 10–15
  MESE, 10–16
  MKDir, 12–17
  Mode, 2–3
  MSI, 12–18
  NAMe, 14–11
  Organization, 5–10
  OUTDrive, 13–10
  OUTPolar, 13–10
  OUTType, 13–11
  PACK, 12–19
  PORTEDGE, 13–12
  PORTLEV, 13–13
  PRINt, 11–10
  PULse, 14–12
  PURGe, 12–20
  REName, 12–22
  RMODe, 10–19
  RTC, 10–20
  SELect, 10–21
  SETColor, 10–23
  SETup, 11–12
  SIGNal, 14–12
  SKEW, 13–14
  STARt, 10–24
  STATEs, 14–14
  STEP, 14–15
  STOP, 10–25
  STORe:CONFig, 12–23
  Structure, 1–5
  SYStem:DATA, 11–5
  SYStem:SETup, 11–12
  TGTctrl, 14–5
  TOGgle, 14–15
  TREE, 13–15
  TYPe, 14–16
  Types, 5–6
  XWINdow, 10–26
Command tree, 5–6
  SELect, 10–22
Common commands, 1–10, 5–6, 9–2
Communication, 1–3
Compound commands, 1–9
Configuration file, 1–4
Control level, 4–4 to 4–5
Controllers, 1–3, 4–5
  mode, 2–3
Conventions, 5–5
COPY, 12–10
CURSTate, 14–9

D
DATA, 11–5
Data
  bits, 3–8
  level, 4–4
  mode, 2–3
  types, 1–13 to 1–14
Data Carrier Detect (DCD), 3–5
DATA command/query, 11–5
Data Communications Equipment,
  see DCE
Data Set Ready (DSR), 3–5
Data Terminal Equipment, 3–3
Data Terminal Ready (DTR), 3–5
DCE, 3–3
DCL, 2–6
DDE, 7–5
Decimal numbers, 1–13
Definite-length block response data, 1–21
DELete, 13–6
Device address, 1–7
  HP-IB, 2–4
  LAN, 4–3
  RS-232-C, 3–9
Device clear, 2–6
Device dependent errors, 8–3
Documentation conventions, 5–5
DOWNload, 12–11
DRIVe, 14–9
DSP, 11–6
DTE, 3–3
Duplicate keywords, 1–10

E
Ellipsis, 5–5
Embedded strings, 1–3, 1–7
Enter statement, 1–3
EOI, 10–13
ERRor, 11–7
Error messages, 8–2
ESB, 7–4
Event Status Register, 7–4
Examples
  BASIC program, 1–19, 12–24, 15–2
  C program, 4–9, 4–11
  RS-232 cables, 3–6
  telnet, 4–12
EXE, 7–5
Execution, 6–3
  errors, 8–4
Exponents, 1–13, 6–9
Extended interface, 3–5
External trigger, 13–12

F
File type numbers, 12–12
Fractional values, 1–14

G
Group execute trigger (GET), 2–6
Group Run, 13–8
  See also Intermodule

H
HEADer, 1–17, 11–8
Headers, 1–7, 1–9, 1–12
Hexadecimal numbers, 1–13
Host language, 1–7
HP-IB, 2–2 to 2–3, 7–8
  address, 2–3
  commands, 7–13
  device address, 2–4
  interface, 2–2 to 2–4
HTIMe, 13–7

I
IEEE 488.1, 2–2, 6–2
  bus commands, 2–6
IEEE 488.2, 6–2
IFC, 2–6
Infinity, 5–4
Initialization, 1–4
INITialize, 12–14
INPort, 13–8
Input buffer, 6–3
INSert, 13–9
Instructions, 1–6
  headers, 1–7
  parameters, 1–8
  syntax, 1–6
  terminator, 1–8
Instrument address, 2–4
Integers, 1–14
Interface capabilities, 2–3
  RS-232-C, 3–8
Interface clear, 2–6
Interface select code
  HP-IB, 2–4
  RS-232-C, 3–9
Intermodule menu, 13–2
  delete module, 13–6
  group run, 13–8
INTermodule subsystem, 13–2
Internal errors, 8–4

K
Keyword data, 1–14
Keywords, 5–3

L
Labels, 1–14
LAN connections, 4–3
  control vs data, 4–4
  mount, 4–4
  net use, 4–4
  socket, 4–11
  telnet, 4–5, 4–12
LASTstate, 14–10
LCL, 7–6
LER, 10–13
Linefeed, 1–8, 5–5
LOAD:CONFig, 12–15
LOAD:IASSembler, 12–16
Local, 2–5
Local lockout, 2–5
LOCKout command, 3–10, 10–14
Longform, 1–12
LONGform command, 1–17, 11–9
Lowercase, 1–12

M
Magic numbers, 12–12
Mainframe commands, 10–2
MAV, 7–4
Measurement complete program, 15–6
Measurement unavailable, 5–4
MENU, 10–15
MESE, 10–16
MESR, 10–18
MKDir, 12–17
MMEMory subsystem, 12–2
Mnemonics, 1–14, 5–3
Module ID numbers, 12–12

Module identification number, 10–9
Mounting, 4–4
MSB, 7–6
MSG, 7–5
MSI, 12–18
MSS, 7–4, 9–16
Msus, 12–2
Multiple
  numeric variables, 1–22
  program commands, 1–15
  queries, 1–22
  subsystems, 1–15

N
NAMe, 14–11
Negative numbers, 1–14
New Line character, 1–8
NL, 1–8, 5–5
Notation conventions, 5–5
Numbers, 1–13, 1–19
  base, 1–20, 1–13
  data, 1–13
  variables, 1–20

O
Octal numbers, 1–13
OPC, 7–5
Operation Complete, 7–6
OR notation, 5–5
OR TRIGGER, 13–8
OUTDrive, 13–10
OUTPolar, 13–10
Output
  buffer, 1–11
  queue, 6–3
OUTPUT statement, 1–3
OUTType, 13–11
Overlapped commands, 5–4, 9–11, 9–19, 10–24 to 10–25

P
PACK, 12–19
Parallel poll, 7–9
  commands, 7–14
Parameters, 1–8
  syntax rules, 1–13
  types, 1–13
Parity, 3–8
Parse tree, 6–8
Parser, 6–3
PON, 7–5
Port In, 13–8, 13–12 to 13–13
Port Out, 13–10 to 13–11, 13–15
PORTEDGE, 13–12
PORTLEV, 13–13
PPC, 7–13
PPD, 7–14
PPE, 7–14
PPU, 7–13
PRINt, 11–10
Printer mode, 2–3
Program examples, 5–12, 15–2
  checking measurement complete, 15–6
  getting ASCII data with PRINt ALL, 15–9
  sending queries to mainframe, 15–7
  SYSTem:SETup, 15–3
  transferring configuration 15–3
Programming
  conventions, 5–5
  message syntax, 1–6
  message terminator, 1–8
Protocol, 3–8, 6–3 to 6–4
  None, 3–8
  XON/XOFF, 3–8
  exceptions, 6–5
PULse, 14–12
PURGe, 12–20

Q
Query, 1–7, 1–11, 1–17
  *ESE, 9–6
  *ESR, 9–7
  *IDN, 9–9
  *IST, 9–9
  *OPC, 9–11
  *OPT, 9–12
  *PRE, 9–13
  *SRE, 9–15
  *STB, 9–16
  *TST, 9–18
  ALL, 14–6
  AUToload, 12–7
  AVAILable, 14–7
  BEEPer, 10–6
  BITS, 14–8
  CAPability, 10–7
  CARDcage, 10–8
  CATalog, 12–8
  CESE, 10–10
  CESR, 10–11
  CURSTate, 14–9
  DATA, 11–6
  EOI, 10–13
  ERRor, 11–7
  FTIMe, 13–7
  HEADer, 11–8
  INPort, 13–8
  LASTstate, 14–10
  LER, 10–13
  LOCKout, 10–14
  LONGform, 11–9
  MENU, 10–16
  MESE, 10–17
  MESR, 10–18
  MSI, 12–18
  NAMe, 14–11
  OUTDrive, 13–10
  OUTPolar, 13–11
  OUTType, 13–11
  PORTEDGE, 13–12
  PRINt, 11–11
  RMODe, 10–19
  SELect, 10–21
  SETColor, 10–24
  SETup, 11–13
  SIGNal, 14–12
  SIGSTatus, 14–13
  SKEW, 13–14
  STATEs, 14–14
  SYSTem:DATA, 11–6
  SYStem:SETup, 11–13
  TREE, 13–16
  TTIMe, 13–17
  TYPe, 14–16
  UPLoad, 12–24
Query errors, 8–5
Query program example, 15–7
Query responses, 1–16, 5–4
Question mark, 1–11
QYE, 7–5

R
Real numbers, 1–14
Real-time clock, 10–20
Receive Data (RD), 3–4 to 3–5
Remote enable (REN), 2–5
REName, 12–22
Request To Send (RTS), 3–5
Response data, 1–21
Responses, 1–17, 5–4
RMODe, 10–19
Root, 5–8
RQC, 7–5
RQS, 7–4, 9–16
RS-232-C, 3–2, 3–9, 6–2
RTC (real-time clock), 10–20

S
Scientific notation, 1–13
SDC, 2–6
SELect command, 10–21
  command tree, 10–22
Selected device clear, 2–6
Sequential commands, 5–4
Serial poll, 7–8
Service Request Enable Register, 7–4
SETColor, 10–23
SETup, 11–12 to 11–13
Shortform, 1–12
SIGNal, 14–12
Signed numbers, 1–14
SIGSTatus, 14–13
Simple commands, 1–9
Skew, 13–7
SKEW command, 13–14
Slot numbers, 10–15
Spaces, 1–8, 1–14, 6–9
Square brackets, 5–5
STARt, 10–24
STATEs, 14–14
Status, 1–23, 7–2, 9–3
  byte, 7–6
  registers, 1–23, 9–3
  reporting, 7–2
STEP, 14–15
Stop bits, 3–8
STOP command, 10–25
STORe:CONFig command, 12–23
String data, 1–14
String variables, 1–19
Subsystem commands, 5–6
  INTermodule, 13–2,
  MMEMory, 12–2
  SYSTem, 11–2
  TGTctrl, 14–2
Suffix multiplier, 6–9
Suffix units, 6–10
Syntax diagrams
  Common commands, 9–4
  IEEE 488.2, 6–5
  INTermodule subsystem, 13–3 to 13–4
  interpretation, 5–4
  Mainframe commands, 10–3 to 10–4
  MMEMory subsystem, 12–3 to 12–4, 12–6
  SYSTem subsystem, 11–3
  TGTctrl subsystem, 14–3 to 14–4
System commands, 5–6
System modules
  talking to, 1–4
SYSTem subsystem, 11–2
SYSTem:SETup program, 15–3

T
Tabs, 1–8
Talk only mode, 2–3
Target Control menu, 14–2
Terminator, 1–8
TGTctrl subsystem, 14–2, 14–5
Three-wire Interface, 3–4
TOGgle, 14–15
Trailing dots, 5–5
Transmit Data (TD), 3–4 to 3–5
TREE command, 13–15
Trouble
  connecting, 4–5
  X Window, 10–26
Truncation rule, 5–3
TTIMe query, 13–17
TYPe, 14–16

U
Units, 1–13, 6–10
UPLoad, 12–24
Uppercase, 1–12
URQ, 7–5

W
White space, 1–8, 6–9

X
XWINdow, 10–26
XXX, 5–5, 5–8
  (meaning of), 1–7

© Copyright Hewlett-Packard Company 1987, 1990, 1993, 1994, 1996
All Rights Reserved.

Reproduction, adaptation, or translation without prior written permission is prohibited, except as allowed under the copyright laws.

Document 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.

Safety

This apparatus has been designed and tested in accordance with IEC Publication 348, Safety Requirements for Measuring Apparatus, and has been supplied in a safe condition. This is a Safety Class I instrument (provided with terminal for protective earthing). Before applying power, verify that the correct safety precautions are taken (see the following warnings). In addition, note the external markings on the instrument that are described under "Safety Symbols."

Warning

- Before turning on the instrument, you must connect the protective earth terminal of the instrument to the protective conductor of the (mains) power cord. The mains plug shall only be inserted in a socket outlet provided with a protective earth contact. You must not negate the protective action by using an extension cord (power cable) without a protective conductor (grounding). Grounding one conductor of a two-conductor outlet is not sufficient protection.
- Only fuses with the required rated current, voltage, and specified type (normal blow, time delay, etc.) should be used. Do not use repaired fuses or short-circuited fuseholders. To do so could cause a shock of fire hazard.
- Service instructions are for trained service personnel. To avoid dangerous electric shock, do not perform any service unless qualified to do so. Do not attempt internal service or adjustment unless another person, capable of rendering first aid and resuscitation, is present.
- If you energize this instrument by an auto transformer (for voltage reduction), make sure the common terminal is connected to the earth terminal of the power source.
- Whenever it is likely that the ground protection is impaired, you must make the instrument inoperative and secure it against any unintended operation.
- Do not operate the instrument in the presence of flammable gasses or fumes. Operation of any electrical instrument in such an environment constitutes a definite safety hazard.
- Do not install substitute parts or perform any unauthorized modification to the instrument.
- Capacitors inside the instrument may retain a charge even if the instrument is disconnected from its source of supply.
- Use caution when exposing or handling the CRT. Handling or replacing the CRT shall be done only by qualified maintenance personnel.

Safety Symbols



Instruction manual symbol: the product is marked with this symbol when it is necessary for you to refer to the instruction manual in order to protect against damage to the product.



Hazardous voltage symbol.



Earth terminal symbol: Used to indicate a circuit common connected to grounded chassis.

WARNING

The Warning sign denotes a hazard. It 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 sign until the indicated conditions are fully understood and met.

CAUTION

The Caution sign denotes a hazard. It 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

Product Warranty

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.

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

Limitation of Warranty

The foregoing warranty shall not apply to defects resulting from improper or inadequate maintenance by 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.

Certification

Hewlett-Packard Company certifies that this product met its published specifications at the time of shipment from the factory. Hewlett-Packard further certifies that its calibration measurements are traceable to the United States National Institute of Standards and Technology, to the extent allowed by the Institute's calibration facility, and to the calibration facilities of other International Standards Organization members.

About this edition

This is the first edition of the *HP 16500C/16501A Programmer's Guide.*

Publication number
16500-97018

Printed in USA.

Edition dates are as follows:

First edition, December 1996

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.

A software or firmware code may be printed before the date. This code indicates the version level of the software or firmware of this product at the time the manual or update was issued. Many product updates 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.

All pages original edition

Artisan Technology Group is an independent supplier of quality pre-owned equipment

Gold-standard solutions

Extend the life of your critical industrial, commercial, and military systems with our superior service and support.

We buy equipment

Planning to upgrade your current equipment? Have surplus equipment taking up shelf space? We'll give it a new home.

Learn more!

Visit us at **artisantg.com** for more info on price quotes, drivers, technical specifications, manuals, and documentation.

Artisan Scientific Corporation dba Artisan Technology Group is not an affiliate, representative, or authorized distributor for any manufacturer listed herein.

We're here to make your life easier. How can we help you today?

(217) 352-9330 | sales@artisantg.com | artisantg.com

ARTISAN® TECHNOLOGY GROUP