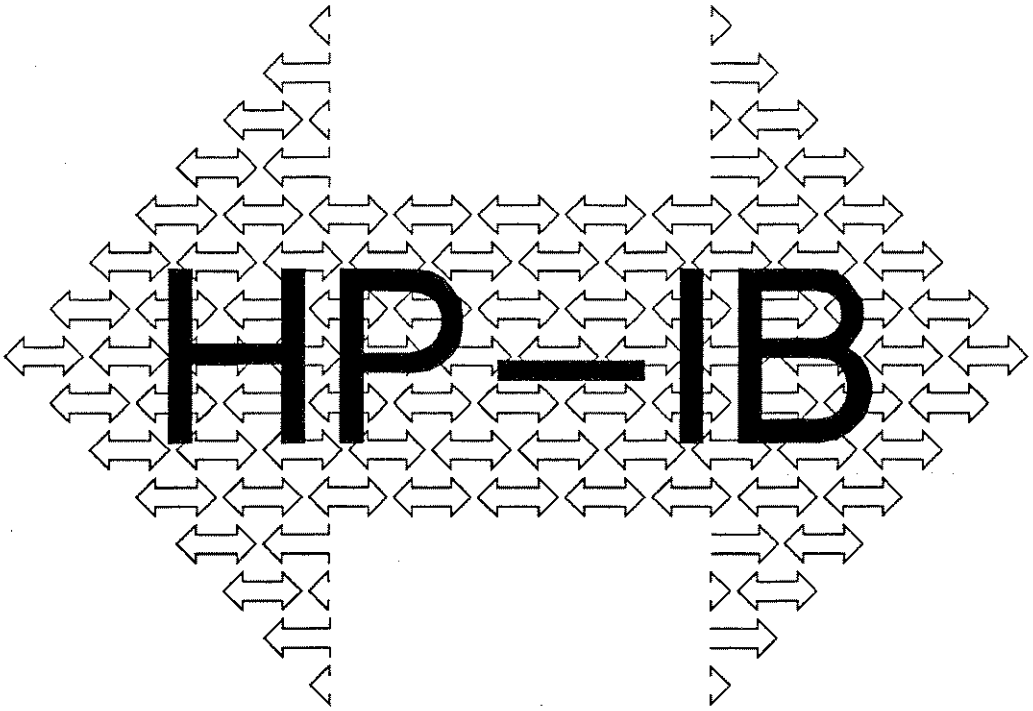
HP-IB Programmer's Guide



HEWLETT PACKARD

HP Part Number 5960-5708
Printed in U.S.A.

Print Date: Sept. 1992

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

In This Book

This book is the *HP-IB Programmer's Guide*. It is intended for people not familiar with HP-IB programming or the programming language, Standard Commands for Programmable Instruments (SCPI).

The *HP-IB Programmer's Guide* introduces the basic concepts of HP-IB programming. Each chapter discusses some aspect of programming the analyzer via the HP-IB:

- Chapter 1 introduces you to remote control of your analyzer via the HP-IB. It also introduces you to SCPI.
- Chapter 2 describes the command hierarchy.
- Chapter 3 tells you how the analyzer interacts with the controller and other devices on the HP-IB.
- Chapter 4 tells you how commands and data are transferred between the analyzer and a controller.
- Chapter 5 describes the analyzer's status groups and tells you how the analyzer uses the status registers to generate service requests.

This book does not contain a detailed description of your analyzer's HP-IB commands or other instrument-specific information. For this information, see your analyzer's *HP-IB Command Reference*.

Table of Contents

3 How the Analyzer Operates in an HP-IB System

4 HP-IB Message Syntax

5 Programming the Status System

1

Remote Control of Your Analyzer

Remote Control of Your Analyzer

Remote control of your analyzer is accomplished via the HP-IB—Hewlett Packard's implementation of the IEEE 488 interface bus.

In a general sense, programming an instrument via HP-IB simply replaces the operation of the instrument from the front panel with commands sent by a computer. For example, instead of using a front-panel key to reset the analyzer, you send the *RST command. The immediate advantage is automation; your computer now controls the analyzer.

The HP-IB interface is a set of signal lines that carry messages between two devices. Functions that are available from the front panel are also available via the HP-IB. In addition, there may be functions available only via the HP-IB. They may allow you to transfer data in and out of the analyzer, write custom signal processing routines, control the display, and communicate with various parts of the instrument.

What is HP-IB?

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

The Hardware

HP-IB cables provide the physical link between devices on the bus. There are eight data lines in each cable that are used to send data from one device to another. Such transfers occur in a byte-serial (one byte at a time), bit-parallel (8 bits at a time) fashion. Devices that can be addressed to send data over these lines are called *talkers*, and those that can be addressed to receive data are called *listeners*. There are also eight control lines in each cable that are used to manage traffic on the data lines and to control other interface operations. Devices that can use these control lines to specify the talker and listener in a data exchange are called *controllers*.

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

HP-IB addresses provide a way to identify devices on the bus. For example, the active controller uses HP-IB addresses to specify which device talks and which device listens during a data exchange. This means that each device's address must be unique.

The device's primary address is set at the factory. It can range from 0 to 30. You can change the default address on the device itself, using a rear-panel switch or a front-panel key sequence.

Sending Commands Over the HP-IB

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

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

Language systems usually deal differently with these two kinds of HP-IB commands. For example, HP BASIC uses a unique keyword to send each bus-management command, but *always* uses the keyword OUTPUT to send device commands. For more information on the differences between bus-management commands and device commands, see chapter 3, "How the Analyzer Operates in an HP-IB System."

How to Use The Examples In This Book

Programming Examples

The semantic requirements of your controller's language determine how the HP-IB commands and responses are handled in your application.

All the programming examples in this book use HP BASIC as the controller language. Since very few specialized HP BASIC commands are used, you can easily modify these examples to work with your computer or other controllers. If you program in another version of BASIC or another language, just substitute your I/O statements for HP BASIC's OUTPUT and ENTER statements.

Command Examples

An example of an HP-IB command looks like this:

```
SOURCE:FREQUENCY:FIXED 1000
```

This example tells you to put the string "SOURCE:FREQUENCY:FIXED 1000" in the output statement appropriate to your application programming language. If you encounter problems, you should investigate the details of how the output statement handles message terminators such as new line (<NL>). If you are using simple OUTPUT statements in HP BASIC, this is taken care of for you. In HP BASIC, you simply type:

```
OUTPUT 711;"SOURCE:FREQUENCY:FIXED 1000"
```

This sends the command to the HP-IB device at address 11 on interface 7.

Most of the command examples in this book do not show message terminators because they are used at the end of every program message and terminators are usually handled by you application programming language. Chapter 4, "Message Syntax" discusses message terminators in more detail.

Response Examples

Response examples look like this:

```
1000
```

These are the characters that you would read from an instrument after sending a query command. To actually pull them from the analyzer into the controller you must use the *input statement* appropriate to your application programming language. If you have problems, you should investigate the details of how the input statement operates. In particular, investigate how the input statement handles punctuation characters such as comma and semicolon, and how it handles message terminators.

To enter the response from a query in HP BASIC, you simply type:

```
OUTPUT 719;"FREQUENCY:SPAN?"

ENTER 719;Span_variable
```

This queries and receives the response from the HP-IB device at address 19 on interface 7.

Response examples do not show response message terminators because they are always <NL><^END>. These terminators are typically handled by the input statement automatically. Chapter 4, "Message Syntax" discusses message terminators in more detail.

Address Examples

The programming examples in this book use "@Analyzer" to represent the device's HP-IB address. For example, the previous query and response example is shown as:

```
OUTPUT @Analyzer;"FREQUENCY:SPAN?"

ENTER @Analyzer;S  pan_variable
```

HP-IB Addressing in HP BASIC

In HP BASIC, the analyzer's address consists of two parts, the device's primary address and an interface select code.

The interface select code, typically 7, indicates which HP-IB port in the system controller is used to communicate with the device.

The interface select code along with the primary address are included in the HP BASIC OUTPUT and ENTER statements.

For example, to select a device on the HP-IB with a select code of 7 and a primary address of 22, you would specify 722.

What is SCPI?

SCPI—the Standard Commands for Programmable Instruments—is a programming language designed specifically for controlling instruments. It defines how to communicate with these instruments from an external controller (computer).

History

Computer-controlled test instruments introduced in the 1960s used a wide variety of non-standard interfaces and communication protocols. During this time, Hewlett-Packard developed the HP-IB as an internal standard. For connectors and cables, HP-IB defined an electrical and mechanical interface. For transmitting data between instruments and computers, it defined handshaking, addressing, and general protocol.

IEEE 488.1

In 1975, the Institute of Electrical and Electronic Engineers (IEEE) approved IEEE 488-1975, which was based on Hewlett-Packard's internal HP-IB standard. They updated this standard to IEEE 488.1-1987. Hewlett-Packard uses *HP-IB* to indicate that an instrument or controller conforms to the IEEE 488.1 standard.

Although it defined how to send bytes of data between instruments and computers, IEEE 488.1 did not specify the data bytes' meanings. Instrument manufacturers freely invented new commands as they developed new instruments. The format of data returned from instruments varied as well. By the early 1980s, work began on additional standards that specified how to interpret data sent via the IEEE 488 bus.

IEEE 488.2

In 1987, the IEEE approved IEEE 488.2-1987. This standard defined the interface capabilities of instruments and controllers in a measurement system connected by the 488 bus (HP-IB). In particular, IEEE 488.2 described how to send commands to instruments and how to send responses to controllers. Although it explicitly defined some frequently used commands, it still left the naming of most commands to instrument manufacturers. This made it possible for similar instruments to conform to IEEE 488.2, yet have entirely different command sets.

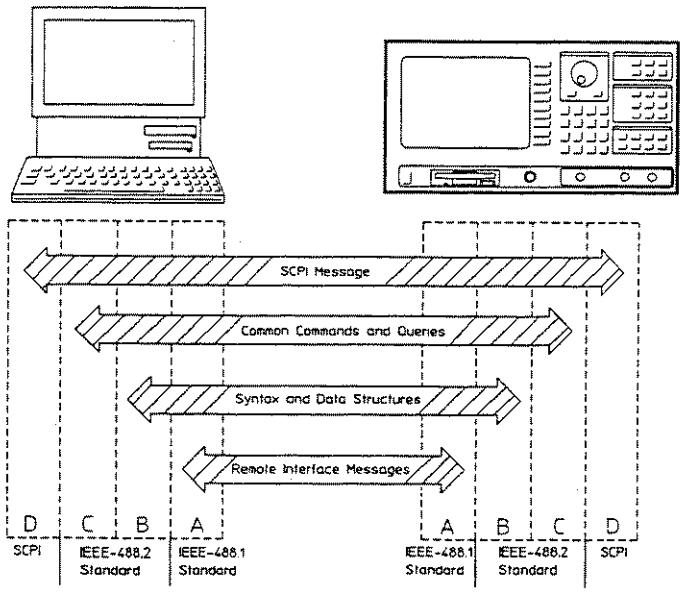
SCPI — A Standard Set of Commands

SCPI goes beyond IEEE 488.2 by defining a standard set of programming commands. For a given measurement function (such as frequency), SCPI defines the specific commands used to access that function via the IEEE 488 bus. If two analyzers conform to the SCPI standard, you would use the same command to set each analyzer's center frequency.

Standard commands provide two advantages:

- If you know how to control functions on one SCPI instrument, you know how to control the same functions on any SCPI instrument.
- Programs written for a particular SCPI instrument are easily adapted to work with a similar SCPI instrument.

The following illustration shows you how SCPI builds on the IEEE 488 standards.



Conceptually it may help to think of these standards as layered, defining different aspects of communication between devices:

- Layer A (IEEE 488.1) defines the physical and electrical connection between devices. It also defines how data is transmitted and how devices are instructed to talk and listen.
- Layer B (IEEE 488.2) defines the syntax and data formats used to send data between devices. It also defines the structure of status registers.
- Layer C (IEEE 488.2) defines the commands used for common tasks (such as resetting the device and reading the Status Byte).
- Layer D (SCPI) defines the commands used to control device-specific functions (such as setting frequency and amplitude). It also defines the parameters accepted by these functions and the values they return.

SCPI and the HP-IB Command Set

Today's HP-IB command set is derived from SCPI. The SCPI command set which is described in the next chapter, differs from the traditional HP-IB command set in the following ways:

- A traditional HP-IB command typically consists of a single mnemonic. A SCPI command typically consists of a series of keywords separated by colons. The keywords are selected from a command hierarchy, which organizes commands into related groups. These multi-keyword commands are less cryptic than single-keyword commands. They can help you make your programs self-documenting. Chapter 2 tells you how to use the command hierarchy.

- A traditional HP-IB command set contains mnemonics that correspond directly to an instrument's front-panel keys. The analyzer's command set gives you HP-IB access to all front-panel functions. However, with SCPI there may not be a one-to-one correspondence between the commands and the front-panel keys. This results from the fact that SCPI command hierarchy is organized differently than the front-panel key hierarchy. Some analyzers have a special feature which allows the analyzer to display equivalent HP-IB command keywords when you press front-panel keys.

SCPI Compliance Information

Many of the HP-IB commands comply with SCPI. The attribute summary in the Command Reference identifies these commands as follows:

- *Confirmed commands* comply with the current version of SCPI.
- *Approved commands* will be added to SCPI in the next revision cycle.
- *Instrument-specific commands* do not comply with SCPI.

Use the SYSTEM:VERSION? query to determine the SCPI version with which your analyzer complies.

To enter the query in HP BASIC, you simply type

```
OUTPUT @Analyzer;"SYSTEM:VERSION?"

ENTER @Analyzer;version
```

The analyzer returns a value that looks like this:

```
1992.0
```

where 1992 is the year-version and 0 is the revision number for that year.

The *HP-IB Command Reference* lists the confirmed and approved commands for your analyzer.

2

Programming with HP-IB Commands

Programming with HP-IB Commands

This chapter describes how to create efficient programs with HP-IB commands. It explains the general structure of the HP-IB command tree. It also explains how to:

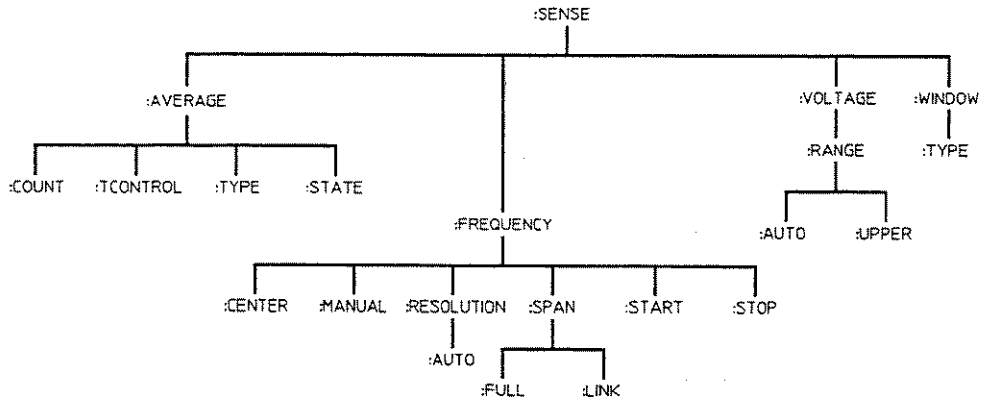
- send multiple commands
- shorten commands by abbreviating keywords
- shorten commands by omitting implied mnemonics

The Command Tree

The HP-IB commands for the analyzer are based on the Standard Commands for Programmable Instruments, known as SCPI. The SCPI standard organizes related instrument functions by grouping them together on a common branch of a command tree. Each branch is assigned a keyword to indicate the nature of the related functions. For example, the analyzer functions that control the presentation of data to the front-panel display are grouped under the DISPLAY branch of the command tree. The DISPLAY branch is only one of the major SCPI branches—called subsystems—used by the analyzer.

When many functions are grouped together in a particular subsystem, additional branching is used to organize these functions into sub-blocks that are even more closely related. The SENSE branch serves as a good example.

The following illustration shows some of the major sub-blocks of the SENSE subsystem. The FREQUENCY functions are grouped under the SENSE branch. The SPAN sub-block groups the commands used to set the frequency span.



The branching process continues until each analyzer function is assigned to its own branch. For example, the function that specifies a flattop window is assigned to the TYPE branch of the WINDOW branch of the SENSE branch. The command looks like this:

```
SENSE:WINDOW:TYPE FLATTOP
```

Note

Colons indicate branching points on the command tree. A parameter is separated from the rest of the command by a space.

Paths Through the Command Tree

To access commands in different paths in the command tree, you need to understand how an instrument interprets commands. A special part of the instrument software, a *parser*, decodes each message sent to the instrument. The parser breaks up the message into component commands using a set of rules to determine which command tree path you are using. The parser keeps track of the *current path*, the level in the command tree where it expects to find the next command you send. This is important because the same keyword may appear in different paths. The particular path you use determines how the keyword is interpreted.

The following rules are used by the parser:

- Power On and Reset — After power is cycled or after *RST, the current path is set to the *root level command*. A root level command is the command closest to the top of the command tree.

- Message Terminators — A message terminator, such as a <NL> character, sets the current path to the root command level. Many programming languages' output statements send message terminators automatically. Message terminators are described in "Program Message Syntax" and "Response Message Syntax" in chapter 4.

- Colon (:) — When a colon is between two command keywords, it moves the current path down one level in the command tree. For example, the colon in SENSE:FREQUENCY specifies that FREQUENCY is one level below SENSE.

When the colon is the first character of a command, it specifies that the next command keyword is a root level command. For example, the colon in :CALCULATE specifies that :CALCULATE is a root level command.

- Semicolon (;) — A semicolon separates two commands in the same message without changing the current path. The following examples specify the type of averaging and turns averaging on. The first requires two program messages; the second requires only one.

```
SENSE:AVERAGE:TYPE RMS
SENSE:AVERAGE:STATE ON

SENSE:AVERAGE:TYPE RMS;STATE ON
```

• <WSP> — Whitespace characters, such as <tab> and <space>, are generally ignored. There are two important exceptions:

Whitespace inside a keyword, such as :CALC ULATE, is not allowed.

You must use white space to separate parameters from commands. For example, the <WSP> between STATE and ON in the command SENSE:AVERAGE:STATE ON is mandatory. Whitespace does not affect the current path.

• Comma (,) — If a command requires more than one parameter, you must separate adjacent parameters using a comma. For example, the SYSTEM:TIME command requires three values to set the analyzer's clock: one for hours, one for minutes, and one for seconds. A message to set the clock to 8:45 AM would be

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

Commas do *not* affect the current path.

• Common Commands — Common commands, such as *RST, are not part of any subsystem. An instrument interprets them in the same way, regardless of the current path setting.

Sending Multiple Commands

You can send multiple commands within a single program message by separating the commands with semicolons. For example, the following program message—sent within an HP BASIC OUTPUT statement—turns on measurement averaging and sets the number of averages to 20:

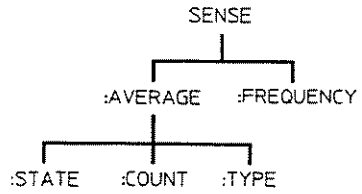
```
OUTPUT @Analyzer;"SENSE:AVERAGE:STATE ON;:SENSE:AVERAGE:COUNT 20"
```

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

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

```
OUTPUT @Analyzer;"SENSE:AVERAGE:STATE ON;COUNT 20"
```

In this version of the program message, the semicolon that separates the two commands is not followed by a colon. Whenever this occurs, the command parser assumes that the keywords of the second command come from the same branch of the tree as the final keyword of the preceding command.



STATE, the final keyword of the preceding command, comes from the AVERAGE branch. So COUNT, the first keyword of the second command, is also assumed to come from the AVERAGE branch.

Command Abbreviation

Each command has a long form and a short form. Only the exact short form or the exact long form are accepted by the analyzer's command parser.

The short forms of the keywords allow you to send abbreviated commands. The keywords' short forms are created according to the following rules:

- If the long form of the command has four or fewer characters, the short form is the same as the long form. For example, ARM remains ARM and COPY remains COPY.
- If the long form of command has more than four characters and the fourth character is a consonant, the short form consists of the first four characters of the long form. For example, CALCULATE becomes CALC.
- If the long form of command has more than four characters and the fourth character is a vowel, the short form consists of the first three characters of the long form. For example, INPUT becomes INP.

Note

The syntax descriptions in the *HP-IB Command Reference* chapters use uppercase characters to identify the short form of a particular keyword. However, the analyzer accepts both lowercase and uppercase characters as equivalent.

If the rules listed in this section are applied to the following program message, the statement

```
OUTPUT @Analyzer;"SENSE:AVERAGE:COUNT 20;TCONTROL EXPONENTIAL;
TYPE RMS;STATE ON"
```

becomes

```
OUTPUT @Analyzer;"SENS:AVER:COUN 20;TCON EXP;TYPE RMS;STAT ON"
```

Forms of HP-IB Commands

There are three forms of HP-IB commands.

- command and query
- command only
- query only

The *HP-IB Command Reference* for your analyzer specifies the form for each command.

Command and Query

These commands can set and query the state of the analyzer. The syntax for the query form appends a question mark (?) to the set form. In most cases, you can query any value that you can set. Therefore, the query form of each command may not be shown explicitly.

For example, the presence of the DISPLAY:ENABLE command implies that a DISPLAY:ENABLE? query also exists.

If you can change the units associated with a set value, you can determine the unit parameter. The unit subsystem has five branches:

- :ANGLE
- :CURRENT
- :POWER
- :TEMPERATURE
- :VOLTAGE

For example, send

```
SENSE:VOLTAGE:LEVEL:UNIT:VOLTAGE?
```

to determine the units associated with the value of the source's voltage level.

Command Only

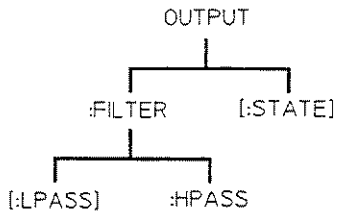
Some commands are events rather than states, so they cannot be queried. An event has no corresponding setting—it causes something to happen inside the analyzer at a particular instant in time. For example, :INITIATE:IMMEDIATE causes a certain trigger event to occur. Because it is an event, there is no query form for the :INITIATE:IMMEDIATE command.

Query Only

There are particular states that the analyzer does not allow you to set. If the command syntax always ends with a question mark (?) in the *HP-IB Command Reference*, it is a *query only* command.

Implied Mnemonics

You can omit some keywords from HP-IB commands without changing the effect of the command. These special keywords are called *implied mnemonics*, and they are used in many subsystems.



The OUTPUT subsystem contains the implied mnemonic STATE at its first branching point. As a result, you can send either of the following commands to the analyzer (using HP BASIC) to turn on the source:

```
OUTPUT @Analyzer;"OUTPUT:STATE ON"
OUTPUT @Analyzer;"OUTPUT ON"
```

The first keyword in the SENSE subsystem is also an implied mnemonic. You can omit it from any SENSE command, even if you are sending a multiple command or if the command parser is down a different path of the tree. These two commands are equivalent

```
OUTPUT @Analyzer;"SENSE:FREQUENCY:SPAN:FULL"
OUTPUT @Analyzer;"FREQUENCY:SPAN:FULL"
```

and so are these

```
OUTPUT @Analyzer;"SENSE:SWEEP:MODE AUTO"
OUTPUT @Analyzer;"SWEEP:MODE AUTO"
```

Note

Implied mnemonics are identified by brackets [ ] in syntax diagrams.

3

How the Analyzer Operates in an HP-IB System

How the Analyzer Operates in an HP-IB System

This chapter explains how the analyzer operates in an HP-IB system. First, it describes the capabilities of the analyzer as the controller of an HP-IB system or as an addressable device on the bus. Next, it explains how the analyzer functions in an HP-IB system and how the analyzer communicates with other devices. Refer to the *HP-IB Command Reference* for a listing of the analyzer's interface capabilities as defined by the IEEE 488.1 standard.

A committee meeting is a good analogy to illustrate how the devices on an HP-IB system interact.

A chairperson directs the activities of a meeting. The chair may recognize a committee member to speak to the other members. The chair may even turn the meeting over to the chair of a subcommittee, to lead the discussion of a specific agenda item.

In an HP-IB system, the *system controller* is the chairperson of the system. It controls all activities on the HP-IB. The system controller uses *addressing* to "recognize" devices of the system. It can address a specific device on the HP-IB to send data to the other devices. In this case, the device sending data is called a *talker* and the devices receiving data are called *listeners*. (In a typical HP-IB system, there is one listener and one talker.) The system controller can give up control of the HP-IB to another device which is then called the *active controller*. The active controller directs the activities of the system until it passes control back to the system controller. The system controller can gain control of the HP-IB at any time, even if it is not the active controller.

Controller Capabilities

There can be only one system controller on the bus. Some analyzers can be configured as an HP-IB system controller. The only time you can configure the analyzer as the system controller is when it is *the only controller on the bus*. Such a setup would be likely if you only wanted to control printers or plotters with the analyzer. It would also be the case if you were using HP Instrument BASIC to control other test equipment.

When you use the analyzer with another controller on the bus, you normally configure it as an addressable-only HP-IB device. In this configuration, the analyzer may be given control so it can be the active controller. It can also function as a talker or listener.

See the *HP-IB Command Reference* to determine how to configure your analyzer.

Command and Data Modes

There are two types of HP-IB commands, those that control the device (device commands) and those that manage the bus (bus-management commands). The HP-IB contains an attention (ATN) line that determines which of these commands can be sent. When the interface is in *command mode* (ATN TRUE), a controller can send bus-management commands over the bus. When the interface is in *data mode* (ATN FALSE), a controller can send device commands and data over the bus.

Device Commands

In data mode, device commands are sent by the controller, but *data can be sent either by the controller or a talker*. The analyzer responds to two different kinds of device commands:

- Common commands, which access device functions required by the IEEE 488.2 standard. All common commands begin with an asterisk (*). They do not change the location of the parser in the command tree.
- Subsystem commands, which access the rest of the analyzer's functions. Most commands are subsystem commands.

The section, "Exchanging Messages," which appears later in this chapter, describes how the analyzer communicates with the controller and other devices on the HP-IB. The analyzer's response to specific device commands are described in the *HP-IB Command Reference* for your analyzer.

Bus-Management Commands

Bus-management commands specify which devices on the interface can talk (send data) and which can listen (receive data). These commands also instruct devices on the bus—either individually or collectively—to perform a particular interface operation. The commands themselves are defined by the IEEE 488.1 standard. Refer to the documentation for your controller's language system to determine how to send these commands. Examples are in HP BASIC.

Device Clear (DCL)

When the analyzer receives this command, it

- clears its input and output queues
- resets its command parser (so it is ready to receive a new program message)
- cancels any pending command or query

The command does not affect the following:

- Front-panel operation.
- Any analyzer operations in progress (other than front-panel operations).
- Any analyzer settings or registers (although clearing the output queue may indirectly affect the Status Byte's Message Available (MAV) bit).

Example

```
CLEAR 7
```

Go To Local (GTL)

This command returns the analyzer to local (front-panel) control. All keys on the analyzer's front-panel are enabled. Normally the front panel is disabled when commands are sent via HP-IB. See the Local Lockout and Remote Enable commands.

Examples

```
LOCAL 711
LOCAL 722
```

Group Execute Trigger (GET)

This command triggers the analyzer (causes it to start collecting measurement data) if the following conditions are true:

- The analyzer is ready to trigger. (Bit 5 of the Operation Status condition register is set to 1.)
- The trigger source is the HP-IB (TRIG:SOUR BUS). Not all analyzers support this configuration.

Examples

```
TRIGGER 712
TRIGGER 719
```

Interface Clear (IFC)

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

Example

```
ABORT 7
```

Local Lockout (LLO)

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

Local lockout ensures that the analyzer's [Local] hardkey is disabled when the analyzer is in the remote mode. When the key is enabled, it allows a front-panel operator to return the analyzer to local mode, thus enabling all other front-panel keys. However, when the key is disabled, it does not allow the front-panel operator to return the analyzer to local mode. The only way to enable the front panel is to send the Go To Local (GTL) command.

Example

```
LOCAL LOCKOUT 7
```

Parallel Poll

Parallel poll, the capability to simultaneously check the status of all instruments on the HP-IB, is optional. If parallel polling is supported, the analyzer recognizes the following bus-management commands:

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

Check the *HP-IB Command Reference* to verify your analyzer has parallel poll capability. If it does not support parallel polling, the analyzer ignores all of the parallel poll commands.

Example

```
PPOLL 7
```

Remote Enable (REN)

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

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

Example

```
REMOTE 7
```

Selected Device Clear (SDC)

The analyzer responds to this command in the same way that it responds to the Device Clear command.

Examples

```
CLEAR 711
CLEAR 719
```

Serial Poll

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

When the Status Byte is returned in response to a serial poll, bit 6 of the Status Byte acts as the Request Service (RQS) bit. If the bit is set to 1, it will be cleared after the Status Byte is returned.

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

Examples

```
SPOLL 714
SPOLL 723
```

Take Control Talker (TCT)

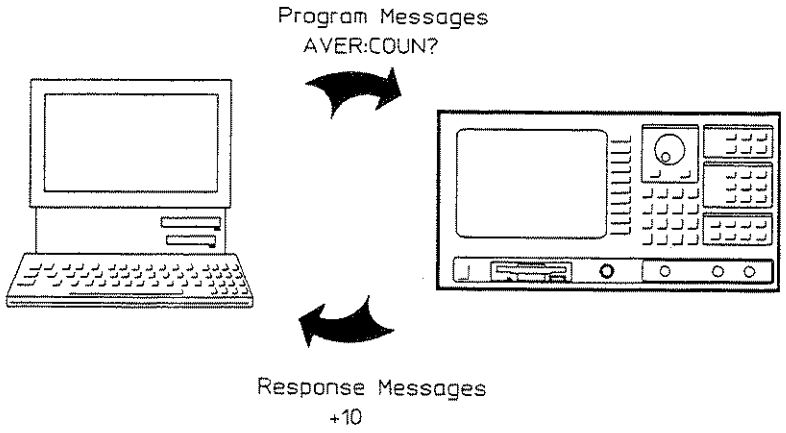
When the analyzer is addressed to talk, this command causes it to take control of the HP-IB. It becomes the active controller on the bus. The analyzer automatically passes control back when it completes the operation that required it to take control. Control is passed back to the address specified by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

If the analyzer does not require control when this command is received, it immediately passes control back. See the example on page 3-28.

Exchanging Messages

The analyzer communicates with the controller and other devices on the HP-IB via program messages and response messages. Program messages are used to send commands, queries, and data to the analyzer. Response messages are used to return data from the analyzer. The syntax for both kinds of messages is presented in chapter 4. Message exchange must conform to the following conventions:

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



HP-IB Queues

Queues enhance the exchange of messages between the analyzer and other devices on the bus. The analyzer contains

- an input queue
- an error queue
- an output queue

Refer to the *HP-IB Command Reference* for instrument-specific queue sizes.

Input Queue

The input queue temporarily stores device commands and queries until they are read by the analyzer's command parser.

The input queue makes it possible for a controller to send multiple program messages to the analyzer without regard to the amount of time required to parse and execute those messages. The queue is cleared when

- you turn on the analyzer
- you send the Device Clear (DCL) or the Selected Device Clear (SDC) command

Error Queue

The error queue temporarily stores error messages. Each time the analyzer detects an error, it places a message in the queue. When you send the SYSTEM:ERROR? query, one message is moved from the error queue to the output queue so it can be read by the controller. Error messages are delivered to the output queue in the order they were received.

The error queue is cleared when

- you turn on the analyzer
- you send the *CLS command

If the error queue overflows, the last error is replaced with error:

```
"Queue Overflow"
```

The oldest errors remain in the queue and the most recent error is discarded.

Output Queue

The output queue temporarily stores a response message until it is read by a controller. It is cleared when

- you turn on the analyzer
- you send the Device Clear (DCL) or the Selected Device Clear (SDC) command
- the analyzer generates a query-interrupted or an unterminated query error

Command Parser

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

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

The parser is reset when

- you turn on the analyzer
- you send the Device Clear (DCL) or the Selected Device Clear (SDC) command
- a colon follows a semicolon in a program message (For more information, see "Sending Multiple Commands" in chapter 2.)
- a program message terminator is received

Query Response Generation

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

- Unterminated condition — This results when you neglect to properly terminate the query with an ASCII line feed character or the HP-IB END message (EOI set true) before you read the response.
- Interrupted condition — This results when you send a second program message before reading the response to the first.
- Buffer deadlock — This results when you send a program message that exceeds the length of the input queue or generates more response data than fits in the output queue. The occurrence of a buffer deadlock is very rare.

Synchronization

It is important that the analyzer is in a known state when a controller sends a device command. It is also important to have a method to determine when a command has finished. Providing for this timing in your program is called command *synchronization*. This section describes when and how to synchronize the analyzer and a controller.

Sequential and Overlapped Commands

The analyzer is capable of processing multiple commands simultaneously. Device commands are executed and processed by the analyzer as fast as they are received. They are *always* executed in the order received.

Device commands can be divided into two broad classes:

- sequential commands
- overlapped commands

Some device commands that you send to the analyzer are processed sequentially. A *sequential command* holds off the processing of subsequent commands until it has been completely processed.

Some commands, called *overlapped commands*, do not hold off the processing of subsequent commands. They allow the execution of subsequent commands while operations initiated by the overlapped command are still in progress.

Typically, overlapped commands take longer to process than sequential commands. For example, the HCOPY:IMMEDIATE command starts the plotting process. The command is not considered to have been completely processed until the plot is complete.

Many commands are overlapped as defined by IEEE 488.2. However, the analyzer's ability to handle multiple operations simultaneously make many of these overlapped commands appear to be sequential commands that do not require synchronization. Example situations where you may need to force synchronization are described in "When to Synchronize" later in the chapter. The Attribute Summary, which appears for each command in the *HP-IB Command Reference*, specifies if synchronization is required.

Synchronization Methods

The analyzer keeps track of overlapped commands. Three commands are available for synchronizing the analyzer with your controller.

- *WAI — Holds off the processing of subsequent commands until the analyzer has completed processing all overlapped commands. However, *it does not stop the controller's operation.* For example, the controller could send a command to another device on the bus. This is the easiest method of synchronization, but it has no impact on the controller.

- *OPC? — Places a 1 in the analyzer's output queue when the analyzer completes processing an overlapped command. The *controller must wait* for the analyzer to complete processing the overlapped command. The HP BASIC command ENTER, reads the output queue. If the result of the query is not read, you get a query interrupt error. Use this query to synchronize your controller to the completion of an overlapped command.

- *OPC — Sets bit 0 of the Standard Event Status event register to 1 when the analyzer completes processing the overlapped command. This generates an interrupt for your controller and requires polling of status bytes or use of the service request (SRQ) capabilities of your controller. (See chapter 5, "Programming the Status System," for more information about the Standard Event Status register set, generating SRQs, and handling interrupts.) Although this command requires more overhead, use it when you need to synchronize your controller to the completion of an overlapped command, but also want to leave the controller free to perform other tasks while the command is processing.

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

In the *HP-IB Command Reference*, the Attribute Summary for each command states if a command requires synchronization (by using the *WAI, *OPC? and *OPC commands).

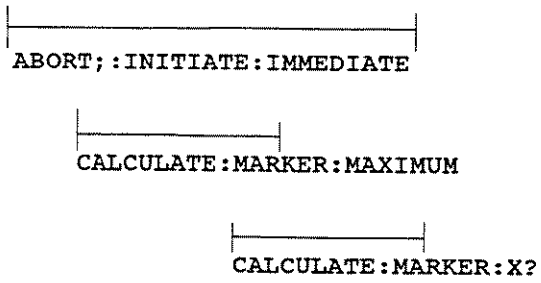
*WAI

This command holds off the processing of subsequent device commands until all overlapped commands are completed. The following example demonstrates the effect of the *WAI command.

Suppose you execute the following series of command to determine which frequency component of a signal contains the greatest amount of energy.

```
OUTPUT @Analyzer;"ABORT;:INITIATE:IMMEDIATE"        !Restart the measurement.
OUTPUT @Analyzer;"CALCULATE:MARKER:MAXIMUM"         !Search for max energy.
OUTPUT @Analyzer;"CALCULATE:MARKER:X?"              !Which frequency?
ENTER @Analyzer; X
PRINT "MARKER at ";X;" Hz"
```

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



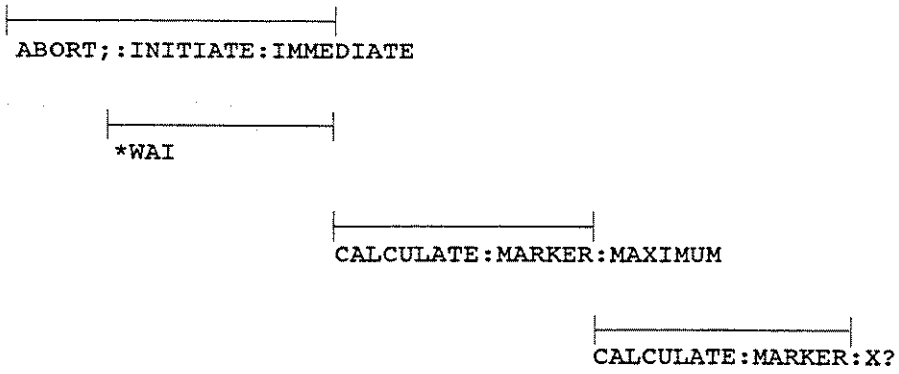
INITIATE:IMMEDIATE is an overlapped command because it does not hold off the processing of the sequential command, CALCULATE:MARKER:MAXIMUM. Remember, INITIATE:IMMEDIATE is not considered complete until the measurement is complete. In this example, the marker searches for maximum energy before the measurement completes. The CALCULATE:MARKER:X? query could return an incorrect value.

To solve the problem, insert a *WAI command.

Example

```
OUTPUT @Analyzer;"ABORT;:INITIATE:IMMEDIATE"        !Restart the measurement.
OUTPUT @Analyzer;"*WAI"                             !Wait until complete.
OUTPUT @Analyzer;"CALCULATE:MARKER:MAXIMUM"         !Search for max energy.
OUTPUT @Analyzer;"CALCULATE:MARKER:X?"              !Which frequency
```

The timeline now looks like this:



The *WAI command keeps the search from taking place until the measurement is completed. The CALCULATE:MARKER:X? query returns the correct value.

Note

Although *WAI stops the analyzer from processing subsequent commands, it does not stop the controller. The controller could send a command to another device on the bus.

*OPC? and *OPC

The *OPC? query pauses the controller until all pending overlapped commands are completed. Design your program so that it must read the analyzer's output queue before the program continues executing.

Example

To determine which frequency component of a signal contains the greatest amount of energy, you would execute the following series of commands:

```
OUTPUT @Analyzer; "ABORT; :INITIATE:IMMEDIATE"      ! Restart the
!                                                     measurement
OUTPUT @Analyzer; "*OPC?"                           ! Wait until complete
ENTER @Analyzer; Meas_done                          ! Read output queue,
!                                                     throw away result
OUTPUT @Analyzer;"CALCULATE:MARKER:MAXIMUM:GLOBAL"  ! Search for max
!                                                     energy
OUTPUT @Analyzer;"CALCULATE:MARKER:X?"              ! Which frequency?
ENTER @Analyzer;Marker_x
PRINT "MARKER at ";Marker_x;" Hz
```

The *OPC command sets bit 0 of the Standard Event register to 1 when all pending overlapped commands are completed. You may poll this bit or you may use the analyzer's register structure to generate a service request (SRQ interrupt). However, your program also must have enabled bit 0 of the Standard Event register and bit 5 of the Status Byte register. (See chapter 5 for more information.) When you synchronize the analyzer and controller in this manner, the controller is free to perform some other task until the service request is generated.

Example

To determine which frequency component of a signal contains the greatest amount of energy, you would execute the following series of commands. The following example uses the polled-bit method.

```
OUTPUT @Analyzer; "ABORT; :INITIATE:IMMEDIATE"      ! Restart the
!                                                     measurement
OUTPUT @Analyzer; "*OPC"                            ! Opc bit set when
!                                                     complete
Start_time=TIMEDATE
REPEAT                                              ! Report elapsed time
!                                                     while waiting
  DISP USING "14A,2D.D";"Elapsed Time: ",
  TIMEDATE-Start_time
  OUTPUT @Analyzer; "*ESR?"                         ! Read the event
!                                                     status register
  ENTER @Analyzer; Esr
  Meas_done=BIT(Esr,0)                              ! Read the operation
!                                                     complete bit
UNTIL (Meas_done)
OUTPUT @Analyzer;"CALCULATE:MARKER:MAXIMUM:GLOBAL"  !Search for max
!                                                     energy
OUTPUT @Analyzer;"CALCULATE:MARKER:X?"              ! Which frequency?
ENTER @Analyzer;Marker_x
PRINT "MARKER at ";Marker_x;" Hz"
```

Example

The following example uses the SRQ interrupt method.

```
OUTPUT @Analyzer;"*CLS"                                    ! Clear the event
!                                                            registers and status
!                                                            byte
OUTPUT @Analyzer;"*ESE 1"                                  ! Standard Event
!                                                            register enable
!                                                            (bit 0=operation complete)
OUTPUT @Analyzer;"*SRE 32"                                 ! Service Request
!                                                            enable
!                                                            (bit 5=standard event)
ENABLE INTR 7;2                                            ! Enable interrupts
!                                                            7 is the interface select
!                                                            code, 2 indicates enabling
!                                                            service request interupt
ON INTR 7,15 RECOVER Meas_done                             ! Interrupt handler
!                                                            7 is the interface select
!                                                            code, 15 indicates highest
!                                                            interupt priority
OUTPUT @Analyzer; "ABORT; :INITIATE:IMMEDIATE"             ! Restart the
!                                                            measurement
OUTPUT @Analyzer; "*OPC"                                   ! Opc bit set when
!                                                            complete
Start_time=TIMEDATE
LOOP                                                       ! Report elapsed time
!                                                            while waiting
  DISP USING "14A,2D.D";"Elapsed Time: ",
  TIMEDATE-Start_time
END LOOP
!                                                            On SRQ interrupt,
!                                                            program execution
!                                                            will jump to this
!                                                            location
Meas_done:
!
OUTPUT @Analyzer;"CALCULATE:MARKER:MAXIMUM:GLOBAL"         ! Search for max
!                                                            energy
OUTPUT @Analyzer;"CALCULATE:MARKER:X?"                     ! Which frequency?
ENTER @Analyzer;Marker_x
PRINT "MARKER at ";Marker_x;" Hz"
```

*OPC only informs you when the pending overlapped commands are completed. *It does not hold off the processing of subsequent commands. As a result, while your program may perform other tasks, you should not send any commands to the analyzer between the time you send* *OPC *and the time you receive the interrupt.* If you send a command during this time it may change the results and would affect how the instrument responds to the previously sent *OPC. If you send commands after an *OPC command, be especially careful to send only commands which do not affect the results of the overlapped command.

When To Synchronize

Although a command may be defined as an overlapped command, *synchronization may not be required.* The multi-tasking operating system of the analyzer allows many overlapped commands to appear sequential. The need to synchronize depends upon the situation in which the overlapped command is executed. The following section describes situations when synchronization is required to ensure a successful operation. The example program segment determines which frequency component of a signal contains the greatest amount of energy.

Completion of a Measurement

To synchronize upon the completion of a measurement, use the ABORT;INITIATE:IMMEDIATE command sequence to initiate the measurement.

This command sequence forces data collection to start (or restart) under the current measurement configuration. A restart sequence, such as ABORT;INITIATE:IMMEDIATE, works just like an overlapped command. It is not considered complete until all operations initiated by that restart command sequence—including the measurement—are finished. The use of the ABORT;INITIATE:IMMEDIATE command sequence together with the *WAI, *OPC? and *OPC commands enable you to determine when a measurement has completed. This ensures that valid measurement data is available for further processing.

Example

The following program segment which uses the *OPC method, ensures a successful operation:

```
                                                    ! Measurement Setup
!                                                     (measurement setup
!                                                     commands would go
!                                                     here)
                                                    ! Start Measurement
OUTPUT @Analyzer; "ABORT; :INITIATE:IMMEDIATE"      ! Restart the
!                                                     measurement
! Synchronize
OUTPUT @Analyzer; "*OPC?"                           ! Wait until complete
ENTER @Analyzer; Meas_done                          ! Read output queue,
!                                                     throw away result
!
! Perform Desired Operation on Measurement Data
!
OUTPUT @Analyzer;"CALCULATE:MARKER:MAXIMUM:GLOBAL"  ! Search for max
!                                                     energy
OUTPUT @Analyzer;"CALCULATE:MARKER:X?"              ! Which frequency?
ENTER @Analyzer;Marker_x
PRINT "MARKER at ";Marker_x;" Hz"
```

Measurements with Manual Arm/HP-IB Trigger

If you are using a manual arm or HP-IB trigger and synchronizing upon the completion of a measurement, an additional programming step is required—supplying the arm or trigger. If you neglect to supply the arm or trigger and attempt to use *WAI or *OPC? to synchronize upon completion of the measurement, your program will never finish. Your program will "hang" or get "stuck." The measurement will not complete until the arm or trigger is received. The analyzer will not accept any more commands over the bus until the measurement is complete.

Example

The following program segment which uses the *WAI method, ensures a successful operation using a manual arm:

```
                                                    ! Measurement Setup
!                                                    (measurement setup
!                                                    commands would go
!                                                    here)
OUTPUT @Analyzer; "ARM:SOURCE MANUAL"
                                                    ! Start Measurement
OUTPUT @Analyzer; "ABORT; :INITIATE:IMMEDIATE"      ! Restart the
!                                                    measurement
                                                    ! Arm
OUTPUT @Analyzer; "ARM:IMMEDIATE"                   ! Manual arm
                                                    ! Synchronize
OUTPUT @Analyzer; "*WAI"                            ! Wait until complete
!   (Note:  OUTPUT @Analyzer;"ABOR; :INIT; :ARM; *WAI" also works.)
!
! Perform Desired Operation on Measurement Data
!
OUTPUT @Analyzer;"CALCULATE:MARKER:MAXIMUM:GLOBAL"  ! Search for max
!                                                    energy
OUTPUT @Analyzer;"CALCULATE:MARKER:X?"              ! Which frequency?
ENTER @Analyzer;Marker_x
PRINT "MARKER at ";Marker_x;" Hz"
```

Example

The following program segment which uses the *OPC polled bit method, ensures a successful operation using an HP-IB trigger:

```
!                                                    ! Measurement Setup
!                                                      (measurement setup
OUTPUT @Analyzer; "TRIGGER:SOURCE BUS"                 commands go here)

OUTPUT @Analyzer; "ABORT; :INITIATE:IMMEDIATE"       ! Start Measurement
!                                                    ! Restart the
                                                       measurement
OUTPUT @Analyzer; "TRIGGER:IMMEDIATE"                ! Trigger
!                                                    ! HP-IB Trigger
!                                                      (Or use:
                                                     OUTPUT @Analyzer; "*TRG")
OUTPUT @Analyzer; "*OPC"                             ! Synchronize
!                                                    ! Opc bit set
!(Note: OUTPUT @Analyzer;"ABOR; :INIT; :TRIG;         when complete
!*OPC" and OUTPUT @Analyzer;"ABOR; :INIT; *TRG;
!*OPC" also work.)
Start_time=TIMEDATE
REPEAT                                               ! Report elapsed time
!                                                      while waiting
  DISP USING "14A,2D.D";"Elapsed Time: ",
  TIMEDATE-Start_time
  OUTPUT @Analyzer; "*ESR?"                          ! Read the event
!                                                      status reg
  ENTER @Analyzer; Esr
  Meas_done=BIT(Esr,0)                               ! Read the
!                                                      operation complete bit
UNTIL (Meas_done)
!
! Perform Desired Operation on Measurement Data
!
OUTPUT @Analyzer;"CALCULATE:MARKER:MAXIMUM:GLOBAL"   ! Search for
!                                                      max energy
OUTPUT @Analyzer;"CALCULATE:MARKER:X?"               ! Which frequency?
ENTER @Analyzer;Marker_x
PRINT "MARKER at ";Marker_x;" Hz"
```

Measurements with External Trigger

If you are using an external trigger, you may use synchronization before the trigger is supplied to the measurement. The program will not "hang" because the trigger signal is applied to a hardware connector on the analyzer and not over the HP-IB.

Example

The following program segment which uses the *OPC SRQ interrupt method, ensures a successful operation using an HP-IB trigger:

```
                                                    ! Measurement Setup
!                                                     (measurement setup
!                                                     commands go here)
OUTPUT @Analyzer; "TRIGGER:SOURCE EXTERNAL"
                                                    ! Set Up For
!                                                     SRQ Interrupt
OUTPUT @Analyzer;"*CLS"                             ! Clear the event
!                                                     registers and
!                                                     status byte
OUTPUT @Analyzer;"*ESE 1"                           ! Standard Event
!                                                     register enable
!                                                     (bit 0=operation complete)
OUTPUT @Analyzer;"*SRE 32"                          ! Service Request enable
!                                                     (bit 5=standard event)
ENABLE INTR 7;2                                     ! Enable interrupts
!                                                     7 = interface select code
!                                                     2 = enabling SRQ interupt
ON INTR 7,15 RECOVER Meas_done                      ! Interrupt handler
!                                                     15 = highest priority
OUTPUT @Analyzer; "ABORT; :INITIATE:IMMEDIATE"      ! Restart the measurement
                                                    ! Synchronize
OUTPUT @Analyzer; "*OPC"                            ! Opc bit set when done
                                                    ! When ext. trigger
!                                                     received and meas done
!                                                     the opc bit is set and
!                                                     interrupt is generated.
Start_time=TIMEDATE
LOOP                                                ! Report elapsed time
!                                                     while waiting
  DISP USING "14A,2D.D";"Elapsed Time: ",TIMEDATE-Start_time
END LOOP
                                                    ! On SRQ interrupt,
!                                                     program execution will
!                                                     jump to this location
!
Meas_done:
!
! Perform Desired Operation on Measurement Data
!
OUTPUT @Analyzer;"CALCULATE:MARKER:MAXIMUM:GLOBAL"  ! Search for max
!                                                     energy
OUTPUT @Analyzer;"CALCULATE:MARKER:X?"              ! Which frequency?
ENTER @Analyzer;Marker_x
PRINT "MARKER at ";Marker_x;" Hz"
```

Averaged Measurements

Averaged measurements work exactly the same way as measurements without averaging with the exception that an averaged measurement is not complete until the average count has been reached. The average count is reached when the specified number of individual measurements has been combined into one averaged measurement result. Synchronization can be used to determine when the average count has been reached.

If the analyzer continues to measure and average the results *after* the average count is reached, synchronization can be used to determine when each subsequent measurement completes.

Synchronization *cannot* be used to determine when each individual measurement completes. To determine if an individual measurement is complete, use the Measuring bit (bit 4) in the Operation Status Group. (Refer to Chapter 5, "Programming the Status System" for more information about the Operation Status Group.)

Example

The following program segment which uses *OPC?, ensures a successful operation in an averaged measurement:

```
                                                    ! Measurement Setup
!                                                     (measurement setup
!                                                     commands would
!                                                     go here)
OUTPUT @Analyzer; "SENSE:AVERAGE:STATE ON"
OUTPUT @Analyzer; "SENSE:AVERAGE:COUNT 10"

                                                    ! Start Measurement
OUTPUT @Analyzer; "ABORT; :INITIATE:IMMEDIATE"      ! Restart the
!                                                     measurement
                                                    ! Synchronize
OUTPUT @Analyzer; "*OPC?"                           ! Wait until complete
ENTER @Analyzer; Avg_done                           ! Read output queue,
!                                                     throw away result
                                                    ! For the next
!                                                     100 averages, store
!                                                     the max energy
!                                                     in an array.
FOR Index=1 TO 100
!
! Perform Desired Operation on Averaged Measurement Data
!
  OUTPUT @Analyzer;"CALCULATE:MARKER:MAXIMUM:GLOBAL"! Search for max
!                                                     energy
  OUTPUT @Analyzer;"CALCULATE:MARKER:X?"          ! Which frequency?
  ENTER @Analyzer;Marker_x(Index)
  OUTPUT @Analyzer; "*OPC?"
  ENTER @Analyzer; Next_meas_done
NEXT Index
```

Arm and trigger conditions must be satisfied for each individual measurement. For measurements without averaging, this requirement can be met simply by following the restart sequence (ABORT;INITIATE:IMMEDIATE) with an ARM or TRIGGER command. In the case of an averaged measurement, the analyzer must be separately armed or triggered for each individual measurement.

Use "Waiting for ARM" (bit 6) and "Waiting for TRIG" bit (bit 5) in the Operation Status Group to determine when the analyzer is ready to be armed or triggered. Arm and trigger signals are ignored if you send them before the analyzer is ready for them.

Example

The following program segment which uses *OPC?, ensures a successful operation in an averaged measurement with manual arming:

```
                                                    ! Measurement Setup
!                                                     (measurement setup
!                                                     commands would go
!                                                     here)
OUTPUT @Analyzer; "ARM:SOURCE MANUAL"
OUTPUT @Analyzer; "SENSE:AVERAGE:STATE ON"
OUTPUT @Analyzer; "SENSE:AVERAGE:COUNT 10"
                                                    ! Start Measurement
OUTPUT @Analyzer; "ABORT; :INITIATE:IMMEDIATE"      ! Restart the
!                                                     measurement
                                                    ! Arm for each
!                                                     average in the
!                                                     average count
FOR Count=1 TO 10
 REPEAT
  OUTPUT @Analyzer;"STATUS:OPERATION:CONDITION?"
  ENTER @Analyzer;Register
  Waiting_for_arm=BIT(Register,6)
 UNTIL (Waiting_for_arm)
 OUTPUT @Analyzer; "ARM:IMMEDIATE"
NEXT Count
                                                    ! Synchronize
OUTPUT @Analyzer; "*OPC?"                           ! Wait until complete
ENTER @Analyzer; Avg_done
!
! Perform Desired Operation on Averaged Measurement Data
!
OUTPUT @Analyzer;"CALCULATE:MARKER:MAXIMUM:GLOBAL"  ! Search for max
!                                                     energy
OUTPUT @Analyzer;"CALCULATE:MARKER:X?"              ! Which frequency?
ENTER @Analyzer;Marker_x
PRINT "MARKER at ";Marker_x;" Hz"
```

Example

The following program segment which uses *WAI, ensures a successful operation in an averaged measurement with HP-IB triggering:

```
                                                    ! Measurement Setup
!                                                     (measurement setup
!                                                     commands would go
!                                                     here)
OUTPUT @Analyzer; "TRIGGER:SOURCE BUS"
OUTPUT @Analyzer; "SENSE:AVERAGE:STATE ON"
OUTPUT @Analyzer; "SENSE:AVERAGE:COUNT 10"
                                                    ! Start Measurement
OUTPUT @Analyzer; "ABORT; :INITIATE:IMMEDIATE"      ! Restart the
!                                                     measurement
                                                    ! Trigger for each
!                                                     average in the
!                                                     average count
FOR Count=1 TO 10
 REPEAT
  OUTPUT @Analyzer;"STATUS:OPERATION:CONDITION?"
  ENTER @Analyzer;Register
  Wait_for_trig=BIT(Register,5)
 UNTIL (Wait_for_trig)
 OUTPUT @Analyzer; "TRIGGER:IMMEDIATE"
NEXT Count
                                                    ! Synchronize
OUTPUT @Analyzer; "*WAI"                            ! Wait until
!                                                     complete
!
! Perform Desired Operation on Averaged Measurement Data
!
OUTPUT @Analyzer;"CALCULATE:MARKER:MAXIMUM:GLOBAL"  ! Search for max
!                                                     energy
OUTPUT @Analyzer;"CALCULATE:MARKER:X?"              ! Which frequency?
ENTER @Analyzer;Marker_x
PRINT "MARKER at ";Marker_x;" Hz"
```

Example

The following program segment which uses *WAI, ensures a successful operation in an averaged measurement with external triggering:

```
                                                    ! Measurement Setup
!                                                     (measurement setup
!                                                     commands would
!                                                     go here)
OUTPUT @Analyzer; "TRIGGER:SOURCE EXTERNAL"
OUTPUT @Analyzer; "SENSE:AVERAGE:STATE ON"
OUTPUT @Analyzer; "SENSE:AVERAGE:COUNT 10"
                                                    ! Start Measurement
OUTPUT @Analyzer; "ABORT; :INITIATE:IMMEDIATE"      ! Restart the
!                                                     measurement
!
                                                    ! Synchronize
OUTPUT @Analyzer; "*WAI"                            ! Wait until complete
                                                    ! When the analyzer
!                                                     has received ten
!                                                     valid external
!                                                     triggers i.e. while
!                                                     the measurement is
!                                                     in a "Waiting for
!                                                     Trigger" state),
!                                                     the analyzer
!                                                     will go on to
!                                                     process the
!                                                     following commands.
!
! Perform Desired Operation on Measurement Data
!
OUTPUT @Analyzer;"CALCULATE:MARKER:MAXIMUM:GLOBAL"  ! Search for max
!                                                     energy
OUTPUT @Analyzer;"CALCULATE:MARKER:X?"              ! Which frequency?
ENTER @Analyzer;Marker_x
PRINT "MARKER at ";Marker_x;" Hz"
```

Restart Sequences

The SCPI defined command sequence for restarting a measurement is ABORT;INITIATE:IMMEDIATE. Your analyzer may have other command sequences that act in the same way—they restart a measurement. If your analyzer has other restart sequences they will work with the synchronization methods just like the ABORT;INITIATE:IMMEDIATE command sequence. You can use *WAI, *OPC or *OPC? to determine when the measurement is complete. The command's description in the *HP-IB Command Reference* specifies if it restarts a measurement.

Example

The following program segment which uses *WAI, illustrates a successful restart operation:

```
OUTPUT @Analyzer; "RESTART"                              ! Restart the measurement
OUTPUT @Analyzer; "*WAI"                                 ! Wait until complete
OUTPUT @Analyzer;"CALCULATE:MARKER:MAXIMUM:GLOBAL"       ! Search for max
!                                                          energy
OUTPUT @Analyzer;"CALCULATE:MARKER:X?"                   ! Which frequency?
ENTER @Analyzer;Marker_x
PRINT "MARKER at ";Marker_x;" Hz"
```

Hardware Setups

If you need to know when a particular analyzer hardware setup command has been completed, you will need to synchronize. For example, if your device-under-test (DUT) is isolated from the analyzer, you may need to know when a change in the source level has been completed before connecting the DUT and proceeding with the measurement sequence. Any of the synchronization methods (*WAI, *OPC, *OPC?) may be used.

Example

The following program segment which uses *OPC?, illustrates a successful hardware setup:

```
OUTPUT @Analyzer; "SOURCE:OUTPUT:STATE ON"       ! Turn on the source
OUTPUT @Analyzer; "SOURCE:POWER -10 DBM"         ! Set the level
OUTPUT @Analyzer; "*OPC?"                        ! Wait until complete
ENTER @Analyzer; Source_ready
OUTPUT @Switch;"A2"                              !Command string
!                                                  to connect DUT
!                                                  to analyzer
```

Passing Control

The analyzer requires temporary control of the HP-IB to complete some commands such as print or plot commands. After sending such a command, the active controller must pass control to the analyzer. When the analyzer completes the command, it automatically passes control of the HP-IB back to the controller. If a command requires control to be passed to the analyzer, a special note appears in the command's description in the *HP-IB Command Reference*. The *HP-IB Command Reference* also contains an example program that passes control to the analyzer.

How to Pass Control

Use the following procedure when passing control to the analyzer:

1. Use the *PCB command to inform the analyzer of the controller's HP-IB address. The analyzer must know the controller's address so it can pass control back. The analyzer automatically passes control of the HP-IB back to the controller when it completes the requested operation.
2. Use the *WAI, *OPC? or *OPC to ensure all pending operations are complete.
3. Enable the analyzer's status registers to generate a service request when the Operation Complete bit is set to 1. (Send *ESE with a value of 1 and *SRE with a value of 32.)
4. Enable the controller to respond to the service request.
5. Send the command that requires control to be passed to the analyzer (for example, the HCOPY command) followed by the *OPC command.
6. Pass control and synchronize the analyzer's operations.

Passing Control and Synchronization

Example

The following HP BASIC program segment which runs on an external controller, uses the *OPC SRQ interrupt method and illustrates a successful pass control and synchronized plot operation:

```
OUTPUT @Analyzer;"*CLS"                          ! Set up SRQ interrupts
!                                                ! Clear the event registers
                                                   and Status Byte
OUTPUT @Analyzer;"*ESE 1"                        ! Standard Event reg enable
!                                                  (bit 0=operation complete)
OUTPUT @Analyzer;"*SRE 32"                       ! Service Request enable
!                                                  bit 5=standard event)
ON INTR 7,15 RECOVER Ready                       ! Interrupt handler
!                                                  7 is the interface select
!                                                  code, 15 indicates highest
!                                                  interupt priority
ENABLE INTR 7;2                                  ! Enable interrupts
!                                                  7 is the interface select
!                                                  code, 2 indicates enabling
!                                                  service request interrupt
                                                 ! Use *WAI, *OPC?,or *OPC to
!                                                  ensure all pending
!                                                  operations are complete
OUTPUT @Analyzer; "*OPC"                         ! Wait for Operation Complete
DISP "WAITING FOR OPERATION COMPLETE"            !  interrupt
LOOP
END LOOP
Ready:                                           !
I=SPOLL(@Analyzer)                               ! Read and clear the
!                                                   Request Service bit in
!                                                   the Status Byte
                                                 ! Set up new interrupt handler
OUTPUT @Analyzer;"*CLS"                          ! Clear the event registers
!                                                   and Status Byte
ON INTR 7,15 RECOVER Hardcopy_done               ! Interrupt handler
ENABLE INTR 7;2
OUTPUT @Analyzer;"*PCB 21"                       ! Tell the analyzer the pass
!                                                  control back address
OUTPUT @Analyzer;"HCOPY"                         ! Tell the analyzer
!                                                  to plot/print
OUTPUT @Analyzer;"*OPC"                          ! Synchronize
PASS CONTROL @Analyzer                           ! Pass control to the analyzer
DISP "WAITING FOR PLOT TO COMPLETE"              ! Wait until control returns
LOOP                                             ! and the plot/print complete
END LOOP
!
Hardcopy_done:                                   !
I=SPOLL(@Analyzer)                               ! Read and clear the
OUTPUT @ANALYZER;"*CLS"                          ! Request Service bit,
!                                                   the event registers,and
!                                                   the Status Byte
DISP "HARD COPY DONE"
END
```

4

HP-IB Message Syntax

HP-IB Message Syntax

As mentioned in chapter 3, the analyzer uses program messages and response messages to communicate with other devices on the HP-IB.

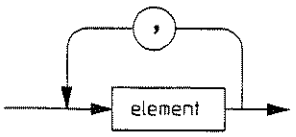
This chapter describes the syntax of HP-IB commands and responses. It provides many examples of the data types used in command parameters and response data. It uses syntax diagrams called *railroad charts* to describe the general syntax rules for both kinds of messages.

- Program Message Syntax explains how to properly construct the messages you send from your computer to your analyzer.
- Response Message Syntax explains the format of messages sent from the analyzer to your computer.
- Data Formats describes the types of data that are contained in program and response messages.

How to Read Railroad Charts

A *railroad chart* is a type of syntax diagram that shows the structure of a programming language. You can use a railroad chart to help you construct a valid message.

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



In the example, the message is composed of an <element> which can repeat if it is separated by a comma.

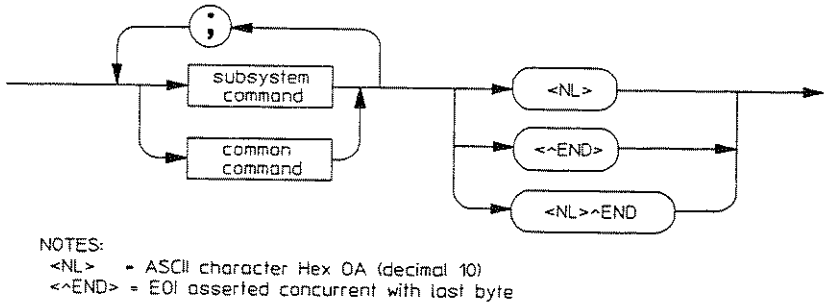
For example, a valid message is:

```
element,element
```

Program Message Syntax

Program messages are the messages that you send from your computer to the analyzer. These program messages contain commands combined with appropriate punctuation and program message terminators. Figure 4-1 illustrates the simplified syntax of a program message.

Figure 4-1. Simplified Program Message Syntax



As figure 4-1 shows, you can send common commands and subsystem commands in the same message. If you send more than one command in the same message, you must separate them with a semicolon. (See "Sending Multiple Commands" in chapter 2.)

You must always end a program message with one of the three program message terminators shown in the illustration. Use <NL>, <^END>, or <NL><^<END> as the *program message terminator*. The word <NL> is an ASCII line feed character. It means new line. The word <^END> means that End or Identify (EOI) is asserted on the HP-IB interface at the same time the preceding data byte is sent. Most programming languages send these terminators automatically. For example, if you use the HP BASIC OUTPUT statement, <NL> is automatically sent after your last data byte. If you are using a PC, you can usually configure your system to send whatever terminator you specify.

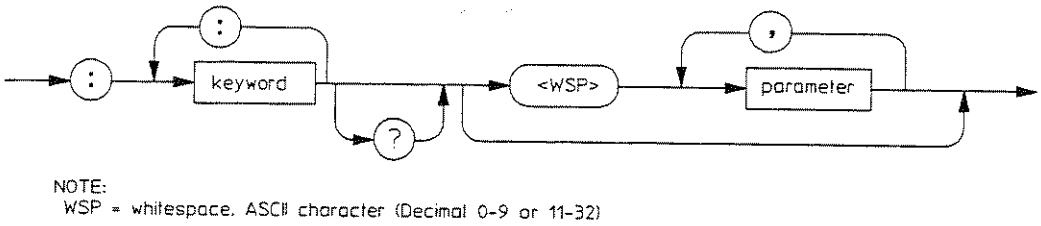
Example of Program Messages

```
ABORT;INITIATE:IMMEDIATE<NL>
*RST;:FREQUENCY:CENTER 50KHZ;SPAN 100KHZ<NL>
TRACE:DATA? D1<^END>
SYSTEM:TIME 15,5,0 <NL><^END>
```

Subsystem Command Syntax

Figure 4-2 describes the basic syntax of subsystems commands.

Figure 4-2. Simplified Subsystem Command Syntax



NOTE:
WSP = whitespace, ASCII character (Decimal 0-9 or 11-32)

As figure 4-2 shows, there must be a whitespace (<WSP>) between the last command keyword and the first parameter in a subsystem command. This is one of the few places where <WSP> is required. If you send more than one parameter with a single command, you must separate adjacent parameters using a comma. Parameter types are explained later in this chapter.

Example of subsystem commands

```
SENSE:AVERAGE:STATE ON
:FREQUENCY:START?
SYSTEM:TIME 15,5,0
```

The *HP-IB Command Reference* includes example program statements for each command.

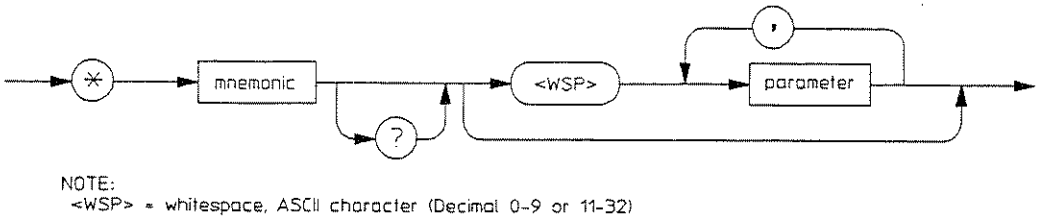
Note

The following program message examples do not show message terminators. They are used at the end of every program message and terminators are usually handled by your application programming language.

Common Command Syntax

Figure 4-3 describes the syntax of common commands.

Figure 4-3. Simplified Command Syntax



NOTE:
<WSP> = whitespace, ASCII character (Decimal 0-9 or 11-32)

As with subsystem commands, you must use a <WSP> to separate a command mnemonic from any subsequent parameters. Adjacent parameters must be separated by a comma. Parameter types are explained later in this chapter.

Common commands do *not* change the location of the parser in the command tree.

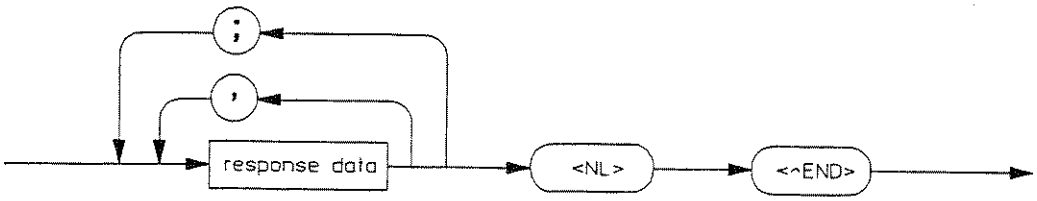
Example of common commands

```
*PCB 11
*SRE 128
*OPT?
```

Response Message Syntax

Response messages are the messages sent from the analyzer to the controller (computer). These messages contain data combined with appropriate punctuation and the message terminators, <NL><^END>. Figure 4-4 shows a simplified view of the response message syntax.

Figure 4-4. Simplified Response Message Syntax



NOTES:
<NL> = newline ASCII character Hex 0A (Decimal 10)
<^END> = EOI asserted concurrent with last byte

Response messages may use comma and semicolons as separators. When a single query command returns multiple values, a comma is used to separate each data item. When multiple queries are sent within the same message, the groups of data items corresponding to each query are separated by a semicolon.

For example, the fictitious query :QUERY1?;QUERY2? might return a response message of:

```
<data1>,<data1>;<data2>,<data2>
```

Response data formats are explained later in this chapter. <NL><^END> is always sent as a response message terminator.

Example of response messages

```
+2.9400000E+04<NL><^END>
+2.9400000E+04;+5.1200000E+04;+3.840000E+03<NL><^END>
+2.9400000E+04;+10;+1<NL><^END>
15,5,0<NL><^END>
```

Note

The response message examples that follow do not show message terminators because they are used at the end of every response message and terminators are usually handled by your application programming language.

Data Formats

Different data formats are used in program messages and response messages. *Parameter data* is data sent in commands from the controller (or computer). *Response data* is data sent from the analyzer to the controller. This accommodates the principle of *forgiving listening* and *precise talking*. Forgiving listening means an instrument is flexible enough to accept commands and parameters in various formats. Precise talking means an instrument always responds to a particular query in a predefined, rigid format. Parameter-data formats are designed to be flexible, in the spirit of forgiving listening. Conversely, response-data formats are defined to meet the requirements of precise talking.

Data Formats

Parameter Formats [1] (controller to analyzer)	**Response-Data Formats [2] (analyzer to controller)**
Numeric	Integer Floating Point
Extended Numeric	Integer Floating Point
Non-decimal Numeric	Hexadecimal Octal Binary
Discrete	Discrete
Boolean	Numeric Boolean
String	String
Expression	String[3]
Block	Definite Length Block Indefinite Length Block

[1] These are flexible formats allowed by forgiving listening.
[2] These are rigid formats required by precise talking.
[3] Expression data can be used as a parameter; however, the data is always returned as a string.

Each parameter format has one or more corresponding response-data formats. For example, a setting that you program using a numeric parameter would return either floating point or integer response data when queried. Whether floating point or integer response data is returned depends on the particular instrument you are using. However, precise talking requires that the response-data format be clearly defined for the analyzer and query. The *HP-IB Command Reference* specifies the data format for individual commands.

Note

If you need more detailed information about a data format refer to the IEEE 488.2 definitions in the *IEEE Standard 488.2-1987, IEEE Standard Codes, Formats Protocols and Common Commands for use with ANSI/IEEE Std 488.1-1987*. Although this document is intended for instrument firmware engineers instead of instrument users and programmers, you may find it useful if you need to know the precise definition of certain message formats, data formats, or common commands.

Parameter Formats

Numeric Parameters

Numeric parameters are used in subsystem commands and common commands. Numeric parameters accept all commonly used decimal representations of numbers including optional signs, decimal points, and scientific notation.

If an instrument setting programmed with a numeric parameter can only assume a finite number of values, then the instrument automatically adjusts the parameter to a legal value. For example, an instrument might have a programmable output impedance of 50 or 75 ohms. If you specified 76.1 for output impedance, it would be rounded to 75. If the instrument setting can only assume integer values, then it automatically rounds the value to an integer. For example, sending *ESE 0.123 has the same result as sending *ESE 0.

Examples of numeric parameters

100.	digits right of decimal are optional
-1.23	leading signs are allowed
4.6e 3	<WSP> allowed after e in exponentials
-7.89E-01	use either E or e in exponentials
+256	leading + is allowed
.5	digits left of decimal point are optional
100	no decimal point is required

Extended Numeric Parameters

Most measurement-related subsystems use extended numeric parameters to specify physical quantities. Extended numeric parameters accept all numeric parameter values and other special values as well. All extended numeric parameters accept MAXimum and MINimum as values. Other special values, such as UP, DOWN, and DEFault may be available as documented in the *HP-IB Command Reference* for your analyzer.

MINimum and MAXimum can be used to set or query values. The query forms are useful for determining the range of values allowed for a given parameter.

Extended numeric parameters accept unit suffixes as part of the parameter value. Acceptable unit suffixes are documented in the *HP-IB Command Reference* for your analyzer.

Note Extended numeric parameters are not used for common commands or STATUS subsystem commands.

Unit Suffix Elements

Class	Preferred Suffix (primary unit)	Allowed Suffix (secondary unit)	Referenced Unit
Amplitude	V		Volt
		VPK	Volts peak
		VRMS	Volts-root-mean-square
Angle	RAD		Radian
		DEG	Degree
Frequency	HZ		Hertz
		MHZ	Megahertz
Power	DBM		Decibel milliwatt
Power	W		Watt
Ratio	DB		Decibel
		DBV	Decibel volt
	PCT		Percent
Resistance	OHM		Ohm
		MOHM	Megohm
Time	S		Second

[1] The suffix units, MHZ and MOHM are special cases which should not be confused with <suffix multiplier>HZ and <suffix multiplier>OHM.

Suffix Multipliers

Unit suffixes can be modified by suffix multipliers.

Value	Name	Mnemonic
1E18	EXA	EX
1E15	PETA	PE
1E12	TERA	T
1E9	GIGA	G
1E6	MEGA	MA[1]
1E3	KILO	K
1E-3	MILLI	M[1]
1E-6	MICRO	U
1E-9	NANO	N
1E-12	PICO	P
1E-15	FEMTO	F
1E-18	ATTO	A

[1] The suffix units, MHZ and MOHM are special cases which should not be confused with <suffix multiplier>HZ and <suffix multiplier>OHM.

Examples of extended numeric parameters

100.	any simple numeric values
-1.23	
4.56e 3	
-7.89E-01	
+256	
.5	
MAX	largest valid setting
MIN	valid setting nearest negative infinity
-100 mV	negative 100 millivolts

Non-decimal Numeric Parameters

In many applications, it is not convenient to transfer data using decimal numbers. *Non-decimal numeric* parameters allow you to specify settings in hexadecimal, octal, or binary formats.

Examples of non-decimal numeric parameters

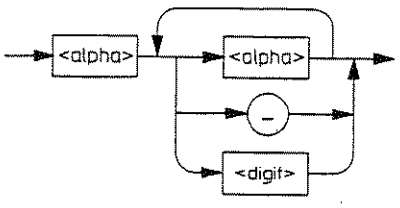
#B0101	binary representation of decimal 5
#Q71	octal representation of decimal 57
#HFA	hexadecimal representation of decimal 250

You may use either upper or lower case letters B, Q, or H to designate the corresponding number base. You may also use upper or lower case letters for the hexadecimal digits A-F.

Discrete Parameters

You use *discrete parameters* to program settings that have a finite number of values. Discrete parameters use keywords to represent each valid setting. They have a long and a short form, just like command keywords.

Discrete Parameter Syntax



Note:
_ = underscore ASCII character SF (Decimal 95)

Examples of discrete parameters

INTERNAL 1	for example, an internal trigger source (channel 1)
EXT	for example, an external trigger source
TIME2	for example, channel 2 time data
NEG	for example, negative edge triggering

Boolean Parameters

Boolean parameters represent a single binary condition that is either true or false. There are only four possible values for a Boolean parameter:

- ON Boolean TRUE
- OFF Boolean FALSE
- 1 Boolean TRUE
- 0 Boolean FALSE

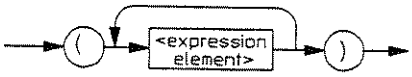
Examples of Boolean parameters

```
SENSE:AVERAGE ON
CALCULATE2:LIMIT:BEEP 0
DISPLAY:ANNOTATION:ALL 1
VOLT:RANG:AUTO OFF
```

Expression Parameters

You can use expressions to set values. The expression must be within parentheses.

Expression Parameter Syntax



Combine the elements according to the rules of algebraic notation and use parentheses to control the order of operations. The *HP-IB Command Reference* specifies the operations and operands available for your analyzer.

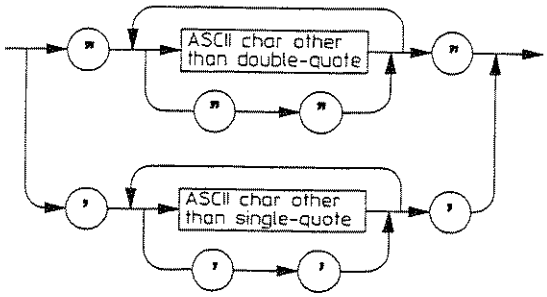
Examples of expression parameters

(PSPEC1*K1)	multiply a power spectrum by a constant
(TIME1 - TIME2)	subtract time data 2 from time data 1
(FRES*K3)	multiply a frequency response by a constant

String Parameters

String parameters can contain virtually any set of ASCII characters, including non-printable characters. A string must begin with a single quote (' ASCII, decimal 39) or a double quote (" ASCII, decimal 34) and end with the same character. The character you chose to mark the beginning and end of the string is called the *delimiter*. You can include the delimiter as part of the string by typing it twice without any characters in between.

String Parameter Syntax



Examples of string parameters

'this is a STRING'

"this is also a string"

"one double quote inside brackets [""]"

'one single quote inside brackets ["]'

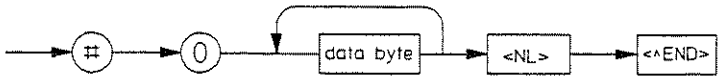
In HP BASIC, it is usually easier to use single quotes. The next command line includes a statement that selects the most recent time record as the measurement data.

```
10 OUTPUT @Analyzer;"CALC:FEED 'XTIM:VOLT'"
```

Block Parameters

Block parameters are typically used to transfer large quantities of related data. Blocks can be sent as *indefinite length blocks* or *definite length blocks*; an analyzer accepts either form.

Indefinite Length Block



Note:
<NL> = newline ASCII character Hex 0A (Decimal 10)

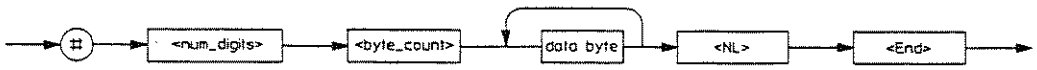
The first two bytes of the data transfer, # and 0, are the header for the block data. The data itself does not begin until the third byte. A mandatory <NL><^END> sequence immediately follows the last byte of block data in an indefinite length block. This forces the termination of the program message.

Example

```
120 OUTPUT @Analyzer;"#0ABC&XYZ" END
```

You must use END to properly terminate the message. In HP BASIC, it asserts an EOI.

Definite Length Block



In the definite length block, two numbers must be specified. The single decimal digit <num_digits> specifies how many digits are contained in <byte_count>. The decimal number <byte_count> specifies how many bytes of data follow in the block.

The elements #, <num_digits>, and <byte_count> make up a header for the block data.

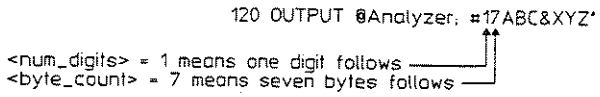
Block Header				Block Data			
byte 1	byte 2	byte 3	byte 4	byte 5	byte 6	...	byte 19
#	2	1	5	<data_byte_1>	<data_byte_2>	...	<data_byte_15>

In the example above, the element <num_digits> is 2. This indicates that the following two bytes are taken together as a single decimal number. In the example, the number is 15. The following 15 bytes are the 5th through 19th bytes of the data transfer. However, they are the 1st through 15th bytes of the data block.

Example

An example HP BASIC statement to send ABC&XYZ as definite length block parameters:

```
120 OUTPUT @Analyzer;"#17ABC&XYZ"
```



Response Data Formats

Response messages are the messages sent from the analyzer to the controller (computer). These messages contain data combined with appropriate punctuation and the message terminators, <NL><^END>.

A large portion of measurement data is formatted as floating-point response data. Floating-point response data are decimal numbers in either fixed decimal notation or scientific notation. In general, you do not need to worry about the rules for formatting floating points and other fixed decimal or scientific notation. Most high-level programming languages that support instrument I/O handle either type transparently.

Examples of floating point response data

1.23E+0

+1.0E+2

0.5E+0

1.23

-100.0

+100.0

Integer Response Data

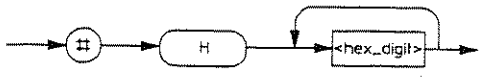
Integer response data are decimal representations of integer values including optional signs. Most status registers related queries return integer response data.

Examples of integer response data

0	signs are optional
+100	leading + sign is allowed
-100	leading sign is allowed
256	never any decimal point

Hexadecimal Response Data

Hexadecimal response data are formatted as base-16 numbers. The H in the hexadecimal response data header, as well as the hexadecimal digits A-F, are always upper case.



NOTE:
<hex_digit> = ASCII character 0-9 and A-F (Decimal 0-15)

Examples of hexadecimal response data

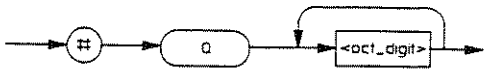
#H0F0F

#H01A1A

#H2B2B

Octal Response Data

Octal response data are formatted as base-8 numbers. The Q in the octal response data header is always upper case.



NOTE:
<oct_digit> = ASCII character 0-7 (Decimal 0-7)

Examples of octal response data

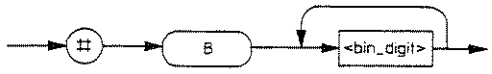
#Q0707

#Q7654

#Q0101

Binary Response Data

Binary response data are formatted as base-2 numbers. The B in the binary response data header is always upper case.



NOTE:
<bin_digit> = ASCII character 0-1 (Decimal 0-1)

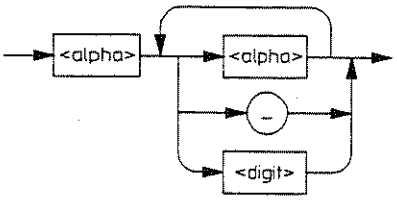
Examples of binary response data

#B00001111

#B00000000

Discrete Response Data

Discrete response data is similar to discrete parameters. The main difference is that discrete response data return only the short form of a particular mnemonic, in all upper case letters.



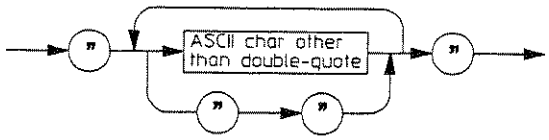
Note:
_ = underscore ASCII character 5F (Decimal 95)

Examples of discrete response data

INT1	for example, a trigger source set to internal (channel 1)
EXT	for example, a trigger source set to external
TIME2	for example, data is channel 2 time data
NEG	for example, a trigger set for negative edges

String Response Data

String response data are similar to string parameters. The main difference is that string response data use only double quotes as delimiters, never single quotes. Embedded double quotes may be present in string response data. Embedded quotes appear as two adjacent double quote marks with no characters in between.



Examples of string response data

"THIS IS VALID"

"SO IS THIS "" "

"I said, ""Hello!"""

Expression Response Data

The analyzer returns expression data surrounded by double-quotes:

Example of expression response data

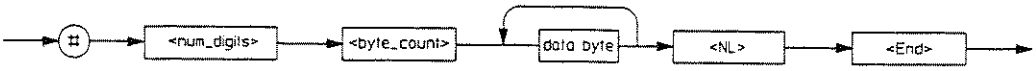
```
"(PSPEC1*K1)"
"(TIME1-TIME2)"
"(FRES*K3)"
```

Definite-Length-Block Response Data

Definite-length-block response data generally contain a large number of related data items, such as measurement trace data. The exact format of data within a block is described for each command in the *HP-IB Command Reference*.

Note

Definite-length-block response data have the same format as definite-length-block parameters.



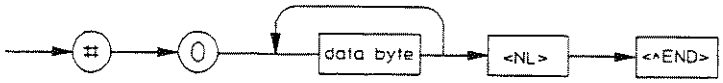
The single decimal digit <num_digits> specifies how many digits are contained in <byte_count>. The decimal number <byte_count> specifies how many bytes of data will follow in <data bytes>.

Examples of definite-length-block response data

#16SAMPLE	6 bytes of data
#2111.1,2.2,3.3	11 bytes of data
#19???+++!!!	9 bytes of data

Indefinite Length Block Response Data

Indefinite length block response data use the same format as indefinite length block parameters.



Note:
<NL> = newline ASCII character Hex 0A (Decimal 10)

A mandatory <NL><^END> sequence immediately follows the last byte of block data in an indefinite length block. This forces the termination of the program message.

Examples of indefinite length block data

```
#0thisisasampleblock<NL><^END>
#001.23,4.56,7.89,9.01<NL>N^END>
#0???++!!!<NL><^END>
#01111111100000000111111111<NL><^END>
```

Example Programs

This section contains three example programs that show you how to transfer trace data between an analyzer and a controller. The example programs are written in HP Instrument BASIC. Explanatory comments follow each program.

Reading Trace Data — ASCII, the first example, shows you how to read trace data values into an array using ASCII representation.

Reading Trace Data — Binary, the second example, shows you how to read floating point data values into an array using binary representation.

Programming the Arbitrary Source, the third example hows you how to load one of the analyzer's data register with data generated on a computer.

Reading Trace Data — ASCII

Example

The following HP Instrument BASIC program reads back the trace data values (401 data points for FFT analysis power spectrums with resolution set to 400) and displays the first ten ASCII values on the HP Instrument BASIC screen.

```
10 DIM A(1:401)
20 ASSIGN @Analyzer TO 800
30 OUTPUT @Analyzer; "FORMAT:DATA ASCII"
40 OUTPUT @Analyzer; "CALC1:DATA?"
50 FOR I=1 TO 401
60  ENTER @Analyzer USING "#,K"; A(I)
70  IF I < 10 THEN PRINT A(I)
80 NEXT I
90 END
```

Example Comments

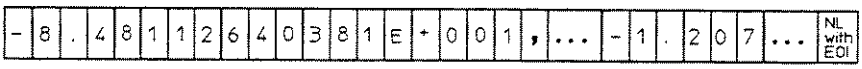
The analyzer generates a 401 data point power spectrum. It was setup to display the power spectrum data on trace A.

10 Dimensions the array for 401 values.

20 Assigns the I/O path @Analyzer to the select code used by HP Instrument BASIC to communicate with the analyzer.

30 The data block format (FORMAT:DATA) specifies extended numeric data with 12 significant digits encoded in ASCII.

40 Query the 401 data points from trace A. This generates the response:



which consists of 401 comma separated numbers terminated with the new line character sent with END. For brevity, only the first and last number are shown.

60 The # in the ENTER statement is an image specifier. It specifies that the statement is automatically terminated as soon as the number is read. The K image specifier reads the number in the default format.

Reading Trace Data — Binary

Example

The following HP Instrument BASIC program reads back the trace data values (401 data points for FFT analysis power spectrums with resolution set to 400 lines) and displays the first ten floating point values on the HP Instrument BASIC screen.

```
  5 DIM A (5000)
 10 ASSIGN @A TO 800
 20 ASSIGN @B TO 800;FORMAT OFF
 30 OUTPUT @A;"FORMAT:DATA REAL,64"
 40 OUTPUT @A;"CALC1:DATA?"
 50 ENTER @A USING "#,A,D";A$,Count
 60 ENTER @A USING "#,"&VAL$(Count)&"D";Num_of_bytes
 70 Num_Points=Num_of_bytes DIV 8
 80 REDIM (Num_Points - 1)
 90 ENTER @B;A(*)
100 ENTER @A;A$
110 FOR I=0 TO 9
120   PRINT A(I)
130 NEXT I
140 END
```

Example Comments

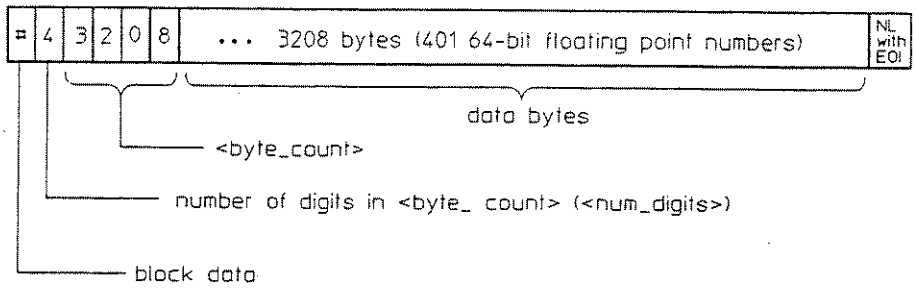
The analyzer generates a 401 data point power spectrum. It was setup to display power spectrum data on trace A.

5 Dimension an array to accomodate a large block of data.

20 Turns the default format off for reading binary data.

30 The data block format (FORMAT:DATA) is 64-bit binary. Binary transfers are much faster than ASCII transfers because the number of data bytes is fewer and the analyzer does not have to construct the extended numeric representation of the numbers from their internal binary representation. The computer does have to translate the extended numeric representation to the computer's binary representation.

40 Query the 401 data points from trace A. This generates the response



where "#" identifies the response as block data, the "4" after the "#" indicates that the next four characters contain the number of data bytes in the data <byte_count>, and the "3208" after the "#4" indicates that there are 3208 data bytes (or 401 64-bit numbers) of data in the block. The last byte of the response is a new line character sent with END.

50 Read the # and <num_digits> — the number of digits contained in the <byte_count>.

60 Read the <byte_count> and assign the value to the variable "Num_of_bytes."

70 Calculate the number of data points for the array. There are 8 data bytes per data point.

80 Change the size of the array s to match the amount of data.

90 Read the 401 numbers in binary.

100 Read the trailing line feed character.

110 - 130 Print the first 10 values.

Programming the Arbitrary Source

Example

The following HP Instrument BASIC program generates and loads a data register with two tone data for use with the arbitrary source in an HP 35665A.

```
  5 DIM D (5000)
 10 ASSIGN @A TO 800
 20 ASSIGN @B TO 800;FORMAT OFF
 30 INTEGER Res,Max_i,T
 40 DIM Tones(0:1)
 50 RAD
 60 Span=102400
 70 Res=400
 80 Tones(0)=7680
 90 Tones(1)=32000
100 SELECT (Res)
110 CASE 100
120 Max_i=255
130 CASE 200
140 Max_i=511
150 CASE 400
160 Max_i=1023
170 CASE 800
180 Max_i=2047
190 CASE ELSE
200 PRINT "illegal measurement resolution"
210 PAUSE
220 END SELECT
230 !
240 REDIM D(Max_i)
250 !
260 Dt=Res/Span/(Max_i+1)
270 FOR T=0 TO 1
280 FOR I=0 TO Max_i
290 D(I)=D(I)+SIN(2*PI*Tones(T)*I*Dt)
300 NEXT I
310 NEXT T
320 !
330 !
340 OUTPUT @A;"FORMAT:DATA REAL,64"
350 OUTPUT @A;"INIT;*WAI;:INIT:CONT OFF"
360 OUTPUT @A;"CALC1:FEED 'XTIM:VOLT 1';*WAI"
370 OUTPUT @A;"TRAC:DATA D1,TRAC1;*WAI"
380 !
390 OUTPUT @A;"TRAC:DATA D1, #0";
400 OUTPUT @B;D(*)
410 OUTPUT @A;CHR$(10) END
420 END
```

Example Comments

The analyzer was setup to display time data on trace A.

5 Dimension an array to accomodate a large block of data.

20 Turn default format off for binary output/enter when using I/O path "B."

30 Define integer variables Res, Max_i, T.

40 Dimension the array for the tone values.

50 Select radians as unit of angular measure.

60 The variable, Span, should be set to the analyzer's current span.

70 The variable, Res, should be set to the analyzer's current resolution. This specific analyzer supports variable resolution of 100, 200, 400 or 800 lines of resolution.

80 - 90 Specify the example tones.

100 - 220 Verifies legal resolution value is selected. An error message appears if an illegal value is selected.

240 Change the size of the array to match the size of the tone data that will be loaded into a data register.

260 Computes the delta t (D) between time points in the tone data.

270 - 310 Compute the tone data for two tones. SIN(2*PI*Tones(T)*I*Dt) generates a time waveform for a tone. The first time through the loop generates time data for the first tone. The second time through the loop, adds in time waveform data for the second tone.

340 - 410 Loads the array into the data register.

340 The data block format (FORMAT:DATA) is 64-bit binary.

350 Pauses the analyzer after one measurement. This helps speed-up the transfer.

360 Display Channel 1 time in trace A.

370 Stores Channel 1 time data in data register 1. Since we saved baseband time data (start frequency is equal to 0) in the data register, the register contains real data.

Before a data register can be loaded with new data from HP-IB, the register must be initialized so it can accept the proper amount and representation of data. To perform this initialization, use the CALC1:FEED command to set the trace data to the type of data you will be putting in the data register then save the trace into the data register using the TRAC:DATA command.

390 TRAC:DATA D1,#0 uses the indefinite length block to load the data into data register 1. The # identifies the data as block data, the "0" after the #" indicates the indefinite length block. The last byte of the output is a new line character sent with END.

400 Output the data in binary.

410 Output the new line.

5

Programming the Status System

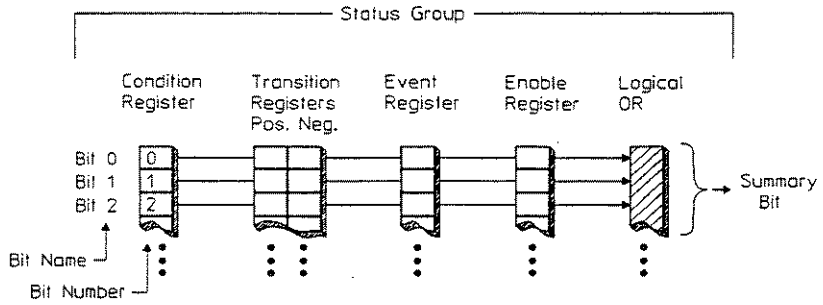
Programming the Status System

The status system contains information about various analyzer conditions. An important feature of SCPI instruments is that they all implement status groups in the same way. This chapter describes the structure of the status system and tells you how to program the status groups.

- The General Status Register Model explains how the status groups are structured in SCPI instruments. It also contains an example of how bits in the various registers change with different input conditions.
- How to Use Registers describes two methods to monitor the registers. It also describes how to respond to service requests.
- The Required Status Groups describes the minimum required status groups present in SCPI instruments. The *HP-IB Command Reference* describes the instrument-specific status groups for your analyzer.

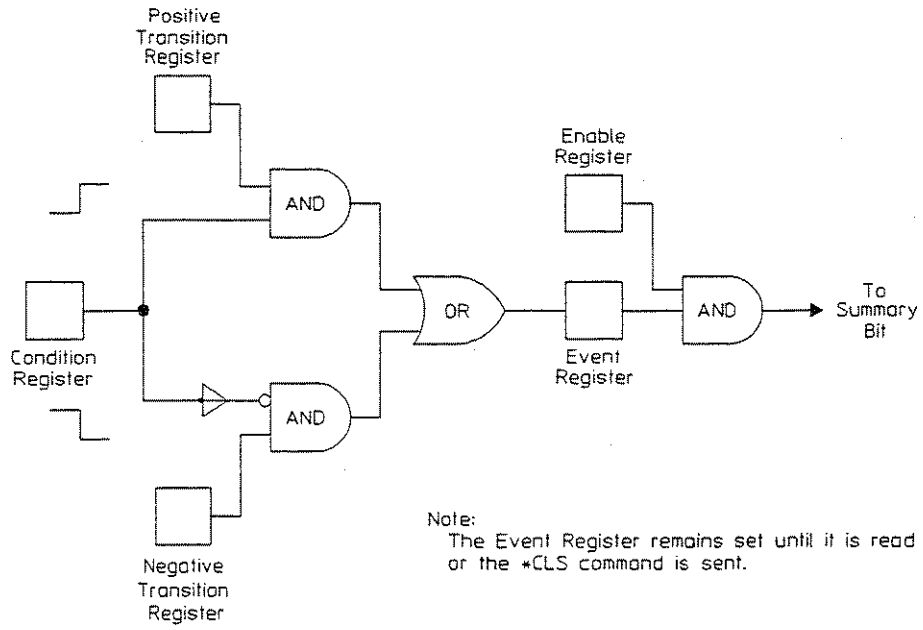
General Status Register Model

The general status register model, shown below is the building block of the analyzer's status system. The model consists of a condition register, an event register, and an enable register. A set of these registers is called a *status group*.



When a status group is implemented in a SCPI instrument, it always contains all of the component registers. However, there is *not* always a corresponding command to read or write to every register.

The flow within a status group starts at the condition register and ends at the register summary bit. (See the illustration below.) You control the flow by altering bits in the transition and enable registers.



Condition Register

The condition register continuously monitors hardware and firmware status. It represents the current state of the instrument. It is updated in real time. When the condition monitored by a particular bit becomes true, the bit is set to 1. When the condition becomes false, the bit is reset to 0. Condition registers are read-only.

If there is no command to read a particular condition register, it is simply invisible to you.

Transition Registers

The transition registers control the reporting of condition changes to the event registers. They specify which types of bit-state changes in the condition register set corresponding bits in the event register. Transition registers are read-write.

Transition register bits may be set for positive transitions, negative transitions, or both. Positive changes in the state of a condition bit (0 to 1) are only reported to the event register if the corresponding positive transition bit is set to 1. Negative changes in the state of a condition bit (1 to 0) are only reported to the event register if the corresponding negative transition bit is set to 1. If you set both transition bits to 1, positive and negative changes are reported to the corresponding event bit.

Transition registers are not affected by *CLS (clear status) or queries. They are set to instrument-dependent values at power-on and after *RST. Some transition registers have a fixed setting if there are no commands to access a particular transition register. This fixed setting along with dependent values are specified in the *HP-IB Command Reference* for your analyzer.

Event Register

The event register records condition changes. When a transition bit allows a condition change to be reported, the corresponding event bit is set to 1. Once set, an event bit is no longer affected by condition changes and subsequent events corresponding to that bit are ignored. The event bit remains set until the event register is cleared— either when the register is read or when the *CLS (clear status) command is sent. Event registers are read-only.

Note

Reading the Event Register, clears the Event Register.

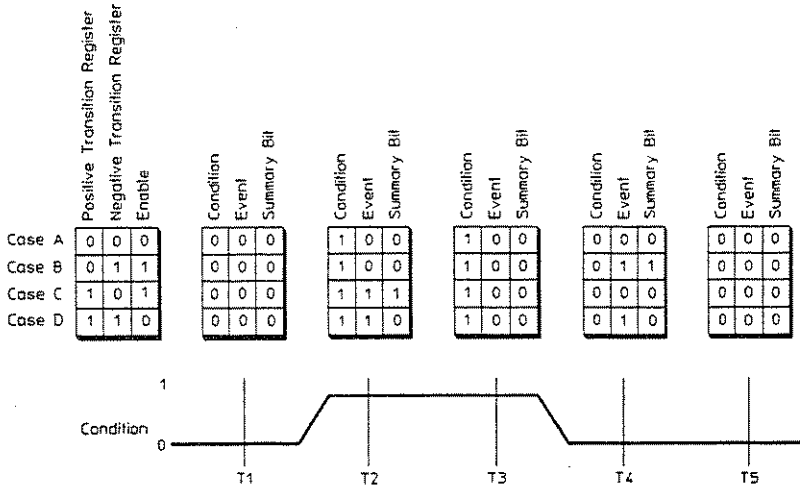
Enable Register

The enable register specifies which bits in the event register set a summary bit to 1. The analyzer logically ANDs corresponding bits in the event and enable registers, and ORs all the resulting bits to determine the state of a summary bit. Summary bits are in turn recorded in another register, often the Status Byte. (The summary bit is only set to 1 if one or more enabled event bits are set to 1.) Enable registers are read-write.

Enable registers are cleared by *CLS (clear status). Querying enable registers does not affect them. There is always a command to read and write to the enable register of a particular status group.

An Example Sequence

The following illustrates the response of a single bit position in a typical status group for various settings. The changing state of the condition in question is shown at the bottom of the figure. A small binary table shows the state of the chosen bit in each status register at selected times (T1 - T5). Each table represents a different situation in relationship to the first table (labeled "Case A", "Case B", "Case C" and "Case D"). Each row contains a variation of the sequence as defined by the settings of the transition and enable registers. The event register is read during each time period. (Remember, reading the event register clears it.)



In cases A and D, the enable bit is zero. The summary bit cannot be set, regardless of the transition register setting. A service request is not made.

In case B, the enable bit and the negative transition register are set to 1. The summary bit is set to 1 when the negative transition occurs during T4. The event register is cleared when it is read during T5.

In case C, the enable bit and the positive transition register are set to 1. The summary bit is set to 1 when the positive transition occurs during T2. The event register is cleared when it is read during T3.

How to Use Registers

There are two methods you can use to access the information in status groups:

- The polling method
- The service request (SRQ) method

Use the polling method when:

- Your language/development environment does not support SRQ interrupts.
- You want to write a simple, single-purpose program and do not want to add the complexity of setting up an SRQ handler.

Use the SRQ method when:

- You need time-critical notification of changes.
- You are monitoring more than one device which supports SRQ.
- You need to have the controller do something else while it is waiting.
- You cannot afford the performance penalty inherent to polling.

The Polling Method

In the polling method, the analyzer has a passive role. It only tells the controller that conditions have changed when the controller asks the right question. In the SRQ method, the analyzer notifies the controller of a condition change without the controller asking. Either method allows you to monitor one or more conditions.

When you monitor a condition with the polling method, you must

1 Determine which register contains the bit that monitors the condition.

2 Send the unique HP-IB query that reads that register.

3 Examine the bit to see if the condition has changed.

The polling method works well if you do not need to know about changes the moment they occur. The SRQ method is more effective if you must know immediately when a condition changes. To detect a change in a condition using the polling method, your program would need to continuously read the registers at very short intervals. This makes the program less efficient. In this case it is better to use the SRQ method.

The SRQ Method

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

1 Determine which bit monitors the condition.

2 Determine how that bit reports to the request service (RQS) bit of the Status Byte.

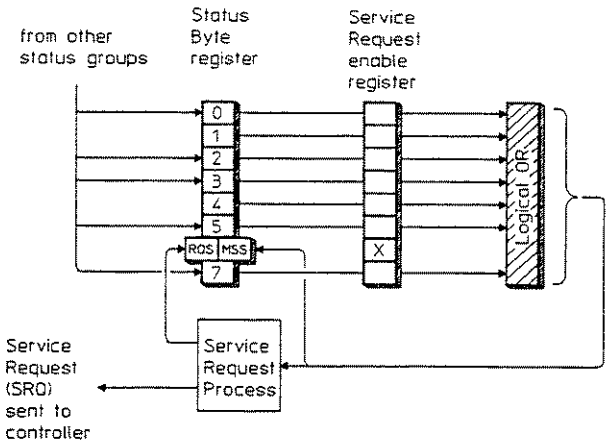
3 Send HP-IB commands to enable the bit that monitors the condition and to enable the summary bits that report the condition to the RQS bit.

4 Enable the controller to respond to service requests.

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

Generating a Service Request

To use the SRQ method, you must understand how service requests are generated. As shown below, other status groups in the analyzer report to the Status Byte. Many of them report directly, but some may report indirectly.



Bit 6 of the Status Byte serves two functions; the request service function (RQS) and the master summary status function (MSS). The RQS bit changes whenever something changes that it is configured to report. The RQS bit is cleared when it is read with a serial poll. The MSS bit is set in the same way as the RQS bit. However, the MSS bit is cleared only when the condition that set it is cleared. The MSS bit is read with *STB?.

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

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

The SRQ process sets the HP-IB's SRQ line true. It also sets the Status Byte's request service (RQS) bit to 1. Both actions are necessary to inform the controller the analyzer requires service. Setting the SRQ line only informs the controller that some device on the bus requires service. Setting the RQS bit allows the controller to determine which device requires service. That is, it tells the controller that this particular device requires service.

If your program enables the controller to detect and respond to service requests, it should instruct the controller to perform a serial poll when the HP-IB's SRQ line is set true. (Refer to "Bus-Management Commands" in chapter 2 for more information about serial polling.) Each device on the bus returns the contents of its Status Byte register in response to this poll. The device whose RQS bit is set to 1 is the device that requested service.

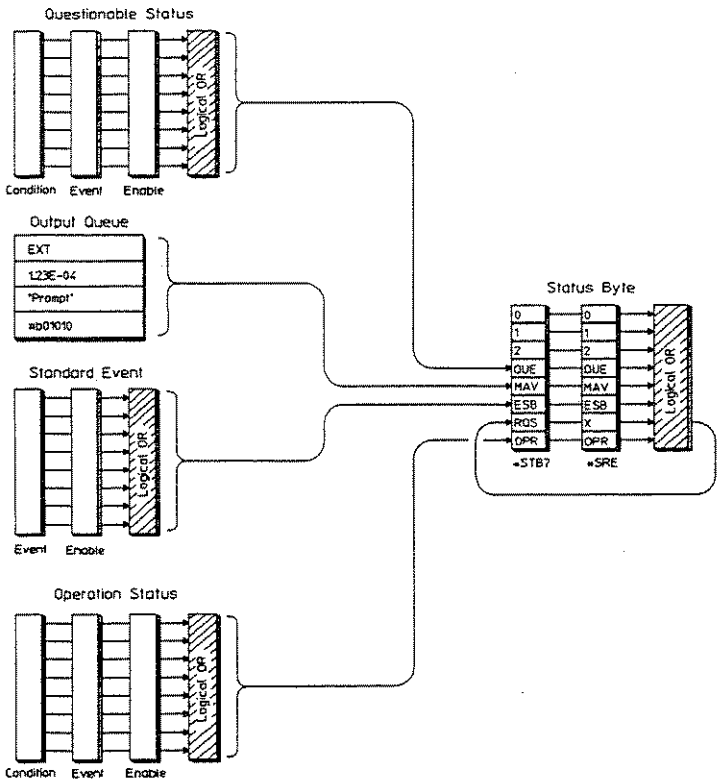
Note

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

Required Status Groups

All SCPI instruments must implement a minimum set of status groups. Additional status groups consistent with the general status register model may also be implemented in your analyzer. They are described in the *HP-IB Command Reference* for your analyzer.

The minimum required status system is shown below.



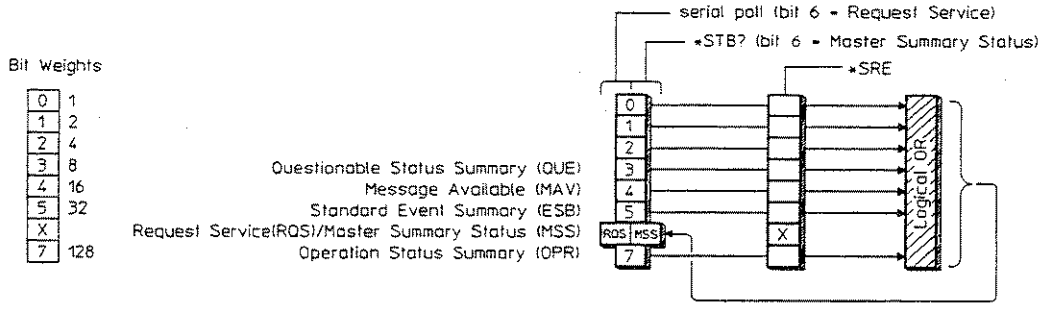
The Operation Status and Questionable Status groups are 16 bits wide, while the Status Byte and Standard Event groups are 8 bits wide. In all 16-bit groups, the most significant bit (bit 15) is not used. Bit 15 is always set to 0.

Note

Register bits not explicitly presented in the following sections may be used by the analyzer. Refer to the *HP-IB Command Reference* for your analyzer.

Status Byte

The Status Byte summarizes the states of the other status groups and monitors the analyzer's output queue. It is also responsible for generating service requests (see "Generating Service Requests" earlier in this chapter).



The Status Byte is unique because it does not exactly conform to the general status model presented earlier. The Status Byte differs from the other groups in the way you read it and how its summary bit is processed.

The Status Byte can be read using either *STB? or a serial poll. Serial poll is a low level HP-IB command that can be executed by SPOLL in HP BASIC. The following HP BASIC code segments are *roughly* equivalent:

Using the *STB? query

```
100  OUTPUT @Analyzer;"*STB?"
110  ENTER @Analyzer;Byte
```

Using a serial poll

```
100 Byte=SPOLL(@Analyzer)
```

The Status Byte summary bit is in bit 6 of the Status Byte. Bit 6 of the Status Byte serves two functions; the request service function (RQS) and the master summary status function (MSS). When bit 6 is set, it generates an SRQ interrupt. This interrupt is a low level HP-IB message that signals the controller that at least one device on the bus requires attention.

There are some subtle differences between *STB? and serial polling. You can use either method to read the state of bits 0-5 and bit 7. However, bit 6 is treated differently depending on whether you use *STB? or serial poll. The RQS bit is read and cleared with a serial poll. The MSS bit is read with *STB?. However, the MSS bit is cleared *only when the condition that set it is cleared*—not when it is read by *STB?.

In general, use serial polling inside interrupt service routines. Whereas *STB? returns the state of the Status Byte, it does not tell you if the analyzer has generated the *current* service request. To ensure the controller can determine which device on the bus caused the interrupt, use serial polling.

Bits in the Status Byte register are set to 1 under the following conditions:

- Questionable Status Summary (bit 3) is set to 1 when one or more enabled bits in the Questionable Status event register are set to 1.
- Message Available (bit 4) is set to 1 when the output queue contains a response message.
- Standard Event Summary (bit 5) is set to 1 when one or more enabled bits in the Standard Event event register are set to 1.
- Master Summary Status (bit 6, when read by *STB?) is set to 1 when one or more enabled bits in the Status Byte register are set to 1.
- Request Service (bit 6, when read by serial poll) is set to 1 by the service request process (see "Generating a Service Request" earlier in this chapter).
- Operation Status Summary (bit 7) is set to 1 when one or more enabled bits in the Operation Status event register are set to 1.

Refer to your analyzer's *HP-IB Command Reference* to determine which bits are used.

The illustration also shows the commands you use to read and write the Status Byte registers. The following statements are example commands using the Status Byte and Status Byte enable register.

***SRE 16** Generate an SRQ interrupt when messages are available in the output queue.

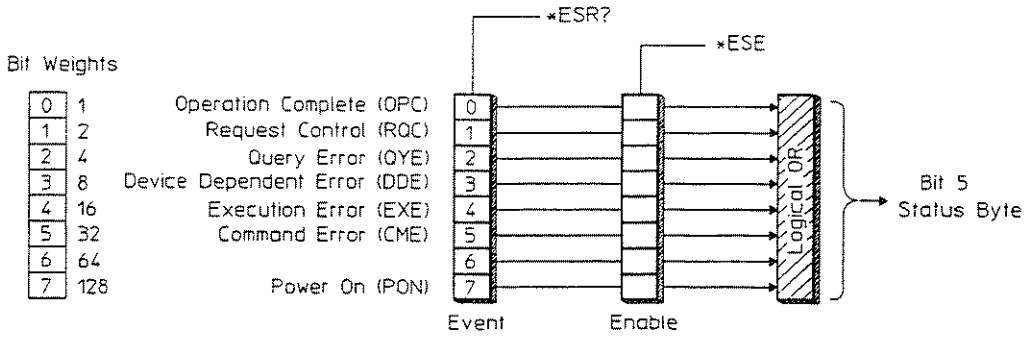
***SRE?** Find out what events are enabled to generated SRQ interrupts.

***STB?** Read the Status Byte event register.

See "Setting and Querying Registers" later in this chapter for more information about these commands.

Standard Event Status Group

The Standard Event status group monitors HP-IB errors and synchronization conditions as shown below. It is one of the simplest and most frequently used. The unique aspect of this group is that you program it using common commands, while you program all other status groups through the STATUS subsystem.



The Standard Event Status group does not conform to the general status register model described at the beginning of this chapter. It contains only two registers: the Standard Event event register and the Standard Event enable register. The Standard Event event register is similar to other event registers, but behaves like a register that has a positive transition register with all bits set to 1. The Standard Event Status enable register is the same as other enable registers.

Bits in the Standard Event Status event register are set to 1 under the following conditions:

- Operation Complete (bit 0) is set to one when the following two events occur (in the order listed):

  – You send the *OPC command to the analyzer.

  – The analyzer completes all pending overlapped commands (see "Synchronization" in chapter 2).

- Request Control (bit 1) is set to 1 when both of the following conditions are true:

  – The analyzer is configured as an addressable-only HP-IB device (see "Controller Capabilities" in chapter 2).

  – The analyzer is instructed to do something (such as plotting or printing) that requires it to take control of the bus.

- Query Error (bit 2) is set to 1 when the analyzer detects a query error.

- Device Dependent Error (bit 3) is set to 1 when the command parser or execution routines detect a device-dependent error.

- Execution Error (bit 4) is set to 1 when the command parser or execution routines detect an execution error.

- Command Error (bit 5) is set to 1 when the command parser detects a command or syntax error.

- Power On (bit 7) is set to 1 when you turn on the analyzer.

Refer to your analyzer's *HP-IB Command Reference* to determine which bits are used.

The illustration also shows the commands you use to read and write the Standard Event status groups. Example commands using Standard Event registers:

***ESE 20** Generate a summary bit whenever there is an execution or command error

***ESE?** Query the state of the standard Event enable register?

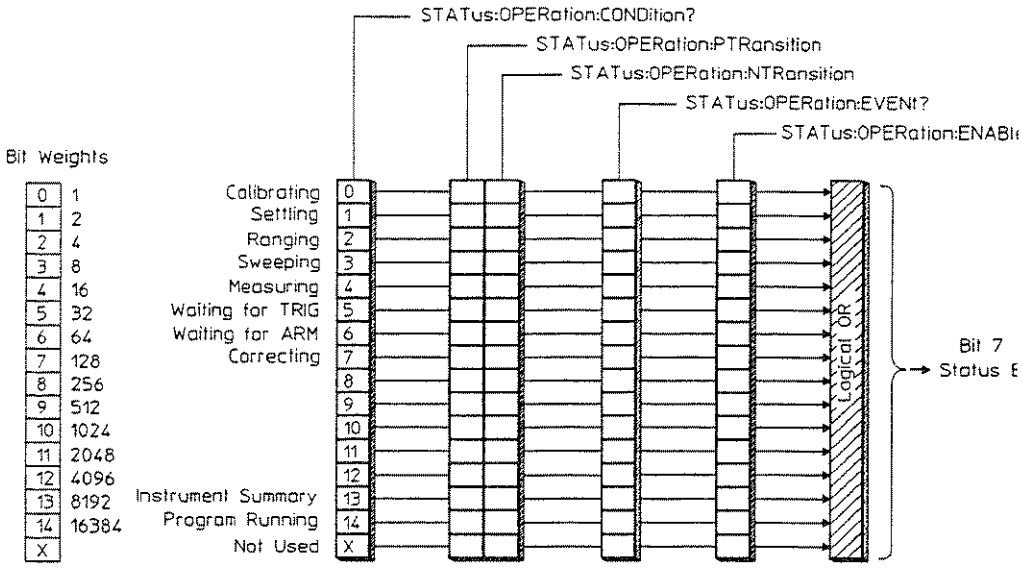
***ESR?** Query the state of the Standard Event event register.

See "Setting and Querying Registers" later in this chapter for more information about using these commands.

Operation Status Group

The Operation Status group monitors conditions in the analyzer's measurement process. It also monitors the state of HP Instrument BASIC programs.

This status group includes a condition register, an event register, and an enable register. It is accessed through the STATUS subsystem. See "Setting and Querying Registers" later in this chapter for more information about using these commands.



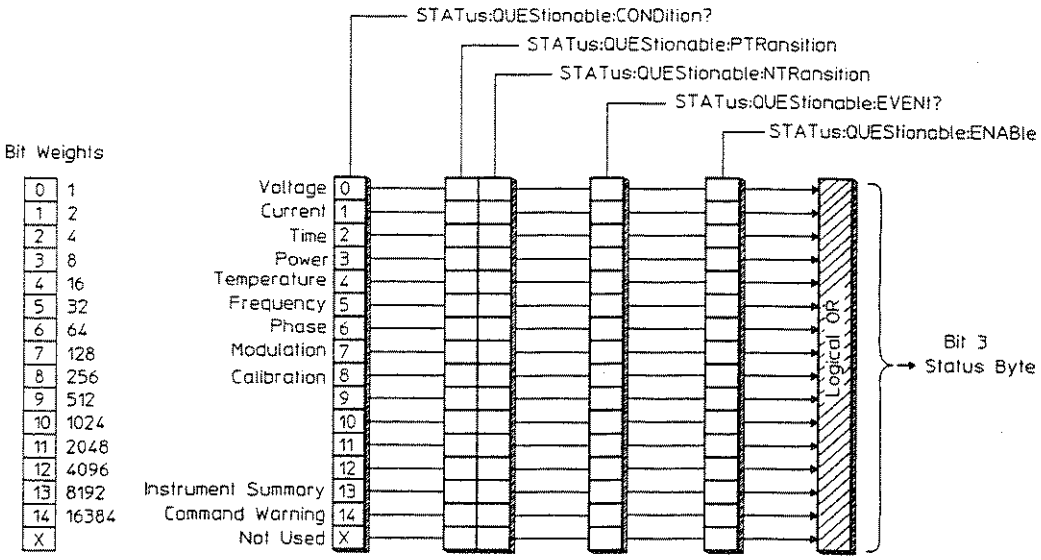
Bits in the Operation Status condition register are set to 1 under the following conditions:

- Calibrating (bit 0) is set to 1 while the self-calibration routine is running.
- Settling (bit 1) is set to 1 while the measurement hardware is settling.
- Ranging (bit 2) is set to 1 while the input range is changing.
- Sweeping (bit 3) is set to 1 while a sweep is in progress.
- Measuring (bit 4) is set to 1 while the analyzer is collecting data for a measurement.
- Waiting for TRIG (bit 5) is set to 1 when the analyzer is ready to accept a trigger signal from one of the trigger sources. (If a trigger signal is sent before this bit is set, the signal is ignored.)
- Waiting for ARM (bit 6) is set to 1 when the analyzer is ready to be armed. If you send the ARM:IMM command before this bit is set, the command is ignored.
- Correcting (bit 7) is set to 1 when the instrument is currently performing a correction.
- Instrument Summary (bit 13) is set to 1 when one of n multiple logical instruments is reporting operational status.
- Program Running (bit 14) is set to 1 while the current HP Instrument BASIC program is running.

Refer to your analyzer's *HP-IB Command Reference* to determine which bits are used.

Questionable Status Group

The Questionable Status register set monitors conditions that affect the quality of measurement data. It also shows the commands you use to read and write the Questionable status group. If you are a beginner, you will rarely need to use this status group.



Bits in the Questionable Status condition register are set to 1 under the following conditions:

- Voltage (bit 0) is set to 1 when the analyzer detects a potential problem with a voltage level.
- Current (bit 1) is set to 1 when the analyzer detects a potential problem with a current level.
- Time (bit 2) is set to 1 when there is questionable time data.
- Power (bit 3) is set to 1 when there is questionable power data.
- Temperature (bit 4) is set to 1 when there is questionable temperature data.
- Frequency (bit 5) is set to 1 when there is questionable frequency data.
- Phase (bit 6) is set to 1 when there is questionable phase data.
- Modulation (bit 7) is set to 1 when there is questionable modulation data.
- Calibration (bit 8) is set to 1 when the last self-calibration attempted by the analyzer failed.
- Instrument Summary (bit 13) is set to 1 when one or more enabled bits in the Questionable Instrument event register are set to 1.
- Command Warning (bit 14) is set to 1 whenever a command such as MEASURE ignores a parameter during execution.

In some analyzers, these bits have another register reporting to them. They are summary bits. For example, if Voltage (bit 0) is a summary bit, it is set to 1 when one or more enabled bits in the Questionable Voltage event register are set to 1.

Refer to your analyzer's *HP-IB Command Reference* to determine which bits are used.

Setting and Querying Registers

The previous status group illustrations include the commands you use to read from and to write to the registers. Most commands have a *set form* and a *query form*.

Use the set form of the command to write to a register. The set form is shown in the illustrations. The set form of a command takes an extended numeric parameter (see "Extended Numeric Parameters" in chapter 4).

Use the query form of the command to read a register. Add a "?" to the set form to create the query form of the command. Commands ending with a "?" in the illustrations are query-only commands. These commands cannot set the bits in the register, they can only query or read the register.

The status group illustrations also include the bit weights you use to specify each bit in the register. For example, to get the Waiting for Trigger condition register (bit 5 in Operation Status group) to generate a service request, send the following commands:

STATUS:PRESET Sets all Enable register bits (*except* the Standard Event and Status Byte registers) to 0. Sets all positive transition bits to 1.

STATUS:OPERATION:ENABLE 32 Sets the Waiting for Trigger Enable register (bit 5) to 1.

***SRE 128** Sets bit 7 of the Service Request Enable register to 1.

See the *HP-IB Command Reference* for more information about these commands.

Example Programs

This section contains two example programs written in HP BASIC.

Responding to an Event Using SRQ, the first example, illustrates how to use the status groups to generate a service request . This also allows a computer to quickly respond to events. The actual measurement reads the Y-marker after each new display is updated.

Trapping Errors Using SRQ, the second example, shows how to use the Standard Event status group and a service request to get a controller to quickly respond to error conditions.

Responding to an Event Using SRQ

Example

This HP BASIC program is an example of the use of using the status groups and the SRQ to allow a controller to quickly respond to events. The measurement reads the Y-marker after each new display is updated.

```
 10  Bus=7
 20 ASSIGN @Analyzer TO 711
 30 DIM Y(1:100)
 40 Readings=0
 50 Max_readings=100
 60 PRINTER IS CRT
 70 OUTPUT @Analyzer;"DISP:ENAB OFF"
 80 OUTPUT @Analyzer;"STAT:PRESET"
 90 OUTPUT @Analyzer;"STAT:DEV:ENABLE 32"
100 OUTPUT @Analyzer;"STAT:DEV:PTR 32"
110 OUTPUT @Analyzer;"*SRE 4   "
120 OUTPUT @Analyzer;"ABORT;:INIT; *WAI"
130 Readings=0
140 !
150 ON INTR Bus GOSUB Record_result
160 ENABLE INTR Bus;2
170 LOOP
180   IF Readings MOD 10=0 THEN DISP Readings
190   EXIT IF Readings>=Max_readings
200 END LOOP
210 PRINT Y(*)
220 STOP
230 Record_result:
240 Serpoll=SPOLL(@Analyzer)
250 IF BIT(Serpoll,2) THEN
260   OUTPUT @Analyzer;"CALC1:MARK:Y?"
270   ENTER @Analyzer;Ymark
280   Readings=Readings+1
290   Y(Readings)=Ymark
300   OUTPUT @Analyzer;"STAT:DEV?"
310   ENTER @Analyzer;Stat
320 END IF
330 IF Readings<Max_readings THEN
340   ENABLE INTR Bus;2
350 END IF
360 RETURN
370 END
```

Example Program Comments

10 - 20 Sets the analyzer's address on the bus.

30 Dimensions the results array.

70 Disables the analyzer's display. This decreases the amount of time it takes to read the marker amplitude value.

80 Clears all registers. (SRQ only happens on 0 to 1 transition.)

90 Enable the Display Ready bit on the Device State status group. (The Device State status group is illustrated below.)

100 Enable on positive transition (0 to 1).

110 Enable SRQ on Device State Register.

120 Start the measurement.

160 The 2 indicates enabling service request interrupt.

240 Read the Status Byte to determine which register caused the SRQ.

250 Continue if Device Status caused the SRQ.

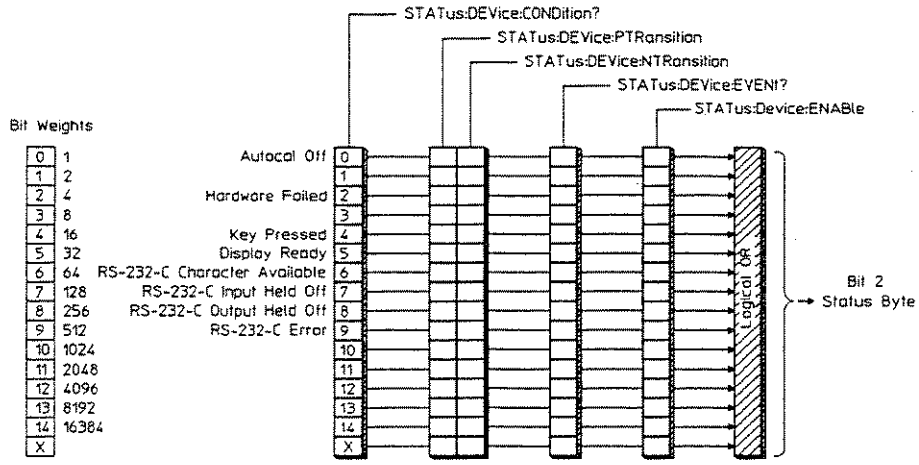
260 Read Marker Amplitude value.

290 Store the Marker Amplitude value into the results array.

300 Clear Device State event register to prepare it for the next transition.

310 "Dummy-read" clears the byte.

340 Re-enable the interrupt. The 2 indicates enabling a service request interrupt.



Trapping Errors Using SRQ

Example

This program is an example of using the Standard Event status group to force an SRQ on an error condition. It allows a controller to quickly respond to error conditions.

```
 10 Bus=7
 20 ASSIGN @Analyzer TO 711
 30 DIM E$[100]
 40 PRINTER IS CRT
 50 !
 60 OUTPUT @Analyzer;"*ESR?"
 70 ENTER @Analyzer;Esr_read
 80 OUTPUT @Analyzer;"*ESE 63"
 90 OUTPUT @Analyzer;"*SRE 32 "
100 OUTPUT @Analyzer;"ABORT;:INIT; *WAI"
110 ON INTR Bus GOSUB Handle_srq
120 ENABLE INTR Bus;2
130 K=0
140 LOOP
150   K=K+1
160   DISP K
170   IF K MOD 100=0 THEN
180   OUTPUT @Analyzer;"INIT: xyzerrorcommand"
190   END IF
200 END LOOP
210 Handle_srq:
220 Serpoll=SPOLL(@Analyzer)
230 IF BIT(Serpoll,5) THEN
240   LOOP
250   OUTPUT @Analyzer;"SYST:ERR?"
260   ENTER @Analyzer;E$
270   EXIT IF VAL(E$)=0
280   PRINT E$
290   END LOOP
300   PRINT ""
310   OUTPUT @Analyzer;"*ESR?"
320   ENTER @Analyzer;Esr_read
330 END IF
340 ENABLE INTR BUS;2
350 RETURN
360 END
```

Example Program Comments

60 Clear the Standard Event Register. (SRQ only happens on 0 to 1 transition.)

70 Perform a read to clear byte.

80 Enable bits 0-5 for miscellaneous errors.

90 Enable SRQ on Standard Event Register.

100 Start the measurement.

120 The 2 indicates enabling a service request interrupt.

180 Every 100th loop forces an error to occur.

220 Read the Status Byte to determine which register caused the SRQ.

230 Continue if the Standard Event register caused the SRQ.

250 - 270 Query the error. In general, multiple reads are used to get the entire stack.

310 Clear the Standard Event register to prepare it for the next transition.

320 Perform a read to clear the byte.

340 Re-enable the interrupt. The 2 indicates enabling a service request interrupt.

Glossary

Active Controller The device currently controlling data exchanges. See also System Controller.

Address A 7-bit code applied to the HP-IB which enables instruments to listen and/or talk on the Bus.

Approved Commands HP-IB commands which will be added to SCPI in the next revision cycle.

ASCII Acronym for American Standard Code for Information Interchange.

ATN A mnemonic for the control line (Attention) which sets the operation of the HP-IB in "Command Mode" (ATN True) or "Data Mode" (ATN False).

Bit A Binary Digit. The smallest part of a binary character which contains intelligible information.

Bit-Parallel Refers to a set of concurrent data bits present on a like number of signal lines used to carry information. Bit-parallel data bits may be acted upon concurrently as a group (byte) or independently as individual data bits.

Bus A set of signal lines used by an interface system to which a number of devices are connected and over which messages are carried.

Bus-Management Commands HP-IB commands which manage the bus specifying which devices on the interface can send data or receive data and instruct devices on the bus to perform an interface operation. The interface must be in "Command Mode."

Byte The binary character sent over the data bus. Although a byte usually refers to 8 bits, frequently the eighth bit is set to 0 in an HP-IB system due to ASCII encoding.

Byte-Serial A sequence of bit-parallel data bytes used to carry information over a common bus.

Command Mode In this mode (ATN True), devices on the HP-IB can be addressed or unaddressed as talkers or listeners. Bus commands are also issued in this mode.

Confirmed Commands HP-IB commands which comply to the current version of SCPI.

Controller Any device that can use the bus' control lines to specify the talker and listener in a data exchange. Any device on the HP-IB which is capable of setting the ATN line and addressing instruments on the Bus as talkers and listeners. (See also System Controller.)

Data Mode In this mode (ATN False), data or instructions are transferred between instruments on the HP-IB.

Definite Length Block Data A data block which takes the following form: #<num_digits><byte_count><data_byte> . . .
where the single decimal digit <num_digits> specifies how many digits are contained in <byte_count>. The decimal number <num_bytes> specifies how many bytes of data follow in the block.

Delimiter A character that separates items of data and can be used to mark the beginning and end of a string.

Device Clear (DCL) ASCII character "DC4" (Decimal 20) which, when sent on the HP-IB in Command Mode returns all devices to their cleared (initialized) state.

Device Commands HP-IB commands which control the device. The interface must be in "data mode."

DIO Mnemonic referring to the HP-IB data lines; DIO1 to DIO8.

EOI Mnemonic referring to the control line "End or Identify" on the HP-IB. This line is used to indicate the end of a multiple byte message on the Bus. It is also used for Parallel Poll.

Forgiving Listening The instrument will accept commands and parameters in various formats.

Go To Local (GTL) ASCII character "SOH" (Decimal 01) which, when sent on the HP-IB in Command Mode, returns devices addressed to listen and capable of responding to local control.

Group Execute Trigger (GET) ASCII character "BS" (Decimal 08) which, when sent on the HP-IB in Command Mode, initiates simultaneous actions by devices addressed to listen and capable of responding to this command.

HP-IB An abbreviation that refers to the "Hewlett-Packard Interface Bus."

IEEE Acronym for Institute For Electrical and Electronic Engineers.

Implied Mnemonic A keyword that you can omit from HP-IB commands without changing the effect of the command.

Indefinite Length Block Data A data block which takes the following form: #0<data_byte> . . . <NL><^END>
where the first two bytes of the data transfer, # and 0, are the header for the block data. The data itself does not begin until the third byte of the data transfer. A mandatory <NL><^END> sequence which immediately follows the last byte of block data forces the termination of the program message.

Instrument Specific Commands HP-IB commands which do not comply to SCPI.

Interface A common boundary between a considered system and another system, or between parts of a system, through which information is conveyed.

Interface System The device-independent mechanical, electrical, and functional elements of an interface necessary to effect communication among a set of devices. Cables, connector, driver and receiver circuits, signal line descriptions, timing and control conventions, and functional logic circuits are typical interface system elements.

Listener A device that can be addressed to receive data over the HP-IB's data lines.

Local Control A method whereby a device is programmable by means of its local (front or rear panel) controls to enable the device to perform different tasks. Also referred to as manual control.

Local Lockout (LLO) An HP-IB multiline universal command (ASCII "DCI" Decimal 17) which disables the return-to-local control on a device (prevents user from leaving remote control other than cycling power). Clearing the REN line of the HP-IB restores local control and re-enables the return-to-local pushbutton on every HP-IB device.

NL Mnemonic for new line or linefeed (ASCII character Hex "0A", Decimal 10).

Non-Decimal Numeric Parameter A parameter which may be specified in hexidecimal, octal or binary formats.

Overlapped Command An HP-IB command which does *not* hold off the processing of subsequent commands. Commands are executed while operations initiated by the overlapped commands are still in progress.

Parallel Poll A method of simultaneously checking the status of instruments on the HP-IB. Each instrument is assigned a DIO line with which to indicate whether it requested service or not. More than one instrument can be connected to one data line.

Parameter Data Data that is sent in an HP-IB command from the controller (computer) to the analyzer.

Parser The part of the analyzer which manages the analyzer's parsing process.

Parsing A process whereby phrases in a string of characters in a computer language are associated with the component names of the grammar that generated that string.

Precise Talking The instrument always responds to a particular query in a predefined, rigid format.

Primary Address That part of the HP-IB address which specifies the device.

Program Messages Send commands, queries and data to the analyzer.

Query Form A form of HP-IB command that reads status registers or the state of the analyzer.

Railroad Chart A syntax diagram which specifies the alternative paths that may be taken in the construction of the allowable structures of a language.

Remote Control A method whereby a device is programmable via its electrical interface connection in order to enable the device to perform different tasks.

REN Mnemonic referring to the control line "Remote Enable" on the HP-IB. This line is used to enable Bus compatible instruments to respond to commands from the controller or another talker. It can be issued only by the system controller.

Response Data Data in program messages that is sent from the analyzer to the controller (computer).

Root Level Command A command closest to the top of the command tree.

SCPI Acronym for the Standard Commands for Programmable Instruments. A standard set of programming commands based on IEEE 488.2.

Select Code That part of the device's address which specifies the interface. Typically, HP-IB select codes are 7 or 8 and are set by the controller's hardware.

Set Form A form of HP-IB command that writes to status registers or sets the analyzer to a specific state.

Sequential Command An HP-IB command which holds-off the processing of subsequent commands until it has been completely processed.

Serial Poll The method of sequentially determining which device connected to the HP-IB has requested service. Only one instrument is checked at a time.

Serial Poll Disable (SPD) ASCII character "EM" (Decimal 25) which, when sent on the HP-IB in Command Mode, causes the bus to leave serial poll mode.

SRQ Mnemonic referring to the control line "Service Request." This control line is used to enable Bus compatible instruments to tell the controller that they require service.

Status Group A set of status registers consisting of an event register, a condition register, transition registers, and an enable register.

Synchronization Procedures placed in a program that allow for the timing of command execution and processing.

System Controller The one device that can take control the bus even if it is not the active controller.

Talker A device that can be addressed to send data over the HP-IB's data lines.

Word A group of bytes treated as a unit and given a single location in memory (organization defines the length of a computer "word") HP computers typically use a word oriented memory with 16-bit (2 byte) words.

WSP Mnemonic referring to white space, ASCII characters Hex "00" to Hex "09" or Hex "0B" to Hex "20" (Decimal 0-9 or 11-32).

Index

D

E

F

G

H

I

K

L

M

N

O

P

Q

R

S

T

U

W