Programmer's Guide

HP 8711C/12C/13C/14C RF Network Analyzers

HP part number: 08712-90057
Printed in USA October 1997 Supersedes: October 1996

Notice

The information contained in this document is subject to change without notice.

Hewlett-Packard makes no warranty of any kind with regard to this material, including but not limited to, the implied warranties of merchantability and fitness for a particular purpose. Hewlett-Packard shall not be liable for errors contained herein or for incidental or consequential damages in connection with the furnishing, performance, or use of this material.

Firmware Revision

This manual documents analyzers with firmware revisions C.04.50 and above. Some features will not be available or will require different keystrokes in analyzers with earlier firmware revisions. For full compatibility, you can upgrade your firmware to the latest version. Contact your nearest Hewlett-Packard sales or service office for information.

HP-IB Programming

This document is an introduction to programming your analyzer over the Hewlett-Packard Interface Bus (HP-IB). Its purpose is to provide concise information about the operation of the instrument under HP-IB control. It provides some background information on the HP-IB and a tutorial introduction using programming examples to demonstrate the remote operation of the analyzer. The examples are provided on two disks that are included with the analyzer. Both disks contain the same examples written mainly in HP BASIC; only the disk format is different. These programs can run on the analyzer's internal controller (Option 1C2) or on an external controller.

- *Example Programs Disk — DOS Format* : part number 08712-10019
- *Example Programs Disk — LIF Format* : part number 08712-10021

You should become familiar with the operation of your network analyzer before controlling it over HP-IB. This document is not intended to teach programming or to discuss HP-IB theory except at an introductory level. Related information can be found in the following references. Contact the nearest HP sales office for ordering information. A list of HP sales and service offices can be found in the "Specifications and Characteristics" chapter of the *User's Guide*.

- Information on making measurements with the analyzer is available in the analyzer's *User's Guide*.
- Information on HP Instrument BASIC is available in the *HP Instrument BASIC User's Handbook*.
- Information on HP BASIC programming is available in the manual set for the BASIC revision being used. For example: *BASIC 7.0 Programming Techniques* and *BASIC 7.0 Language Reference*.
- Information on using the HP-IB is available in the *Tutorial Description of the Hewlett-Packard Interface Bus* (HP literature no. 5021-1927).

Contents

Figures

Tables

Contents

1

Introduction to HP-IB Programming

Introduction to HP-IB Programming

HP-IB — the Hewlett-Packard Interface Bus — is a high-performance bus that allows individual instruments and computers to be combined into integrated test systems. The bus and its associated interface operations are defined by the IEEE 488.1 standard. The IEEE 488.2 standard defines the interface capabilities of instruments and controllers in a measurement system, including some frequently used commands.

HP-IB cables provide the physical link between devices on the bus. There are eight data lines on each cable that are used to send data from one device to another. Devices that send data over these lines are called **Talkers**. **Listeners** are devices that receive data over the same lines. There are also five control lines on each cable that are used to manage traffic on the data lines and to control other interface operations. **Controllers** are devices that use these control lines to specify the talker and listener in a data exchange. When an HP-IB system contains more that 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 one device that can take control of the bus even if it is not the active controller. The network analyzer can 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. 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. A device's address is set on the device itself, using either a front-panel key sequence or a rear-panel switch.

To set the HP-IB address on the analyzer use the softkeys located in the (SYSTEM OPTIONS) HP-IB menu. The factory default address for the analyzer is 16.

NOTE

Throughout this manual, the following conventions are used:

Square brackets ([ ]) are used to enclose a keyword that is optional or implied when programming the command; that is, the instrument will process the command to have the same effect whether the option node is omitted or not.

Parameter types (< >) are distinguished by enclosing the type name in angle brackets.

A vertical bar (|) can be read as "or" and is used to separate alternative parameter options.

Bus Structure

Data Bus

The data bus consists of eight lines that are used to transfer data from one device to another. Programming commands and data sent on these lines is typically encoded in the ASCII format, although binary encoding is often used to speed up the transfer of large arrays. Both ASCII and binary data formats are available to the analyzer. In addition, every byte transferred over HP-IB undergoes a **handshake** to ensure valid data.

Handshake Lines

A three-line handshake scheme coordinates the transfer of data between talkers and listeners. This technique forces data transfers to occur at the speed of the slowest device, and ensures data integrity in multiple listener transfers. With most computing controllers and instruments, the handshake is performed automatically, which makes it transparent to the programmer.

Control Lines

The data bus also has five control lines that the controller uses both to send bus commands and to address devices:

IFC — Interface Clear. Only the system controller uses this line. When this line is true (low) all devices (addressed or not) unaddress and go to an idle state.

ATN — Attention. The active controller uses this line to define whether the information on the data bus is a **command** or is **data**. When this line is true (low) the bus is in the command mode and the data lines carry bus commands. When this line is false (high) the bus is in the data mode and the data lines carry device-dependent instructions or data.

SRQ Service Request. This line is set true (low) when a device requests service: the active controller services the requesting device. The analyzer can be enabled to pull the SRQ line for a variety of reasons.

REN Remote Enable. Only the system controller uses this line. When this line is set true (low) the bus is in the remote mode and devices are addressed either to listen or talk. When the bus is in remote and a device is addressed, the device receives instructions from HP-IB rather than from its front panel (pressing the `Return to Local` softkey returns the device to front panel operation). When this line is set false (high) the bus and all devices return to local operation.

EOI End or Identify. This line is used by a talker to indicate the last data byte in a multiple byte transmission, or by an active controller to initiate a parallel poll sequence. The analyzer recognizes the EOI line as a terminator and it pulls the EOI line with the last byte of a message output (data, markers, plots, prints, error messages). The analyzer does not respond to parallel poll.

Sending Commands

Commands are sent over the HP-IB via a controller's language system, such as IBASIC, QuickBASIC or C. The keywords used by a controller to send HP-IB commands vary among systems. When determining the correct keywords to use, keep in mind that there are two different kinds of HP-IB commands:

- Bus management commands, which control the HP-IB interface.
- Device commands, which 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.

The following example shows how to send a typical device command:

```
OUTPUT 716;"CALCULATE:MARKER:MAXIMUM"
```

This sends the command within the quotes (`CALCULATE:MARKER:MAXIMUM`) to the HP-IB device at address 716. If the device is an analyzer, the command instructs the analyzer to set a marker to the maximum point on the data trace.

HP-IB Requirements

Number of Interconnected Devices:	15 maximum
Interconnection Path/Maximum Cable Length:	20 meters maximum or 2 meters per device, whichever is less.
Message Transfer Scheme:	Byte serial/ bit parallel asynchronous data transfer using a 3-line handshake system.
Data Rate:	Maximum of 1 megabyte per second over limited distances with tri-state drivers. Actual data rate depends on the transfer rate of the slowest device involved.
Address Capability:	Primary addresses: 31 talk, 31 listen. A maximum of 1 talker and 14 listeners at one time.
Multiple Controller Capability:	In systems with more than one controller (like the analyzer system), only one can be active at a time. The active controller can pass control to another controller, but only the system controller can assume unconditional control. Only one system controller is allowed. The system controller is hard-wired to assume bus control after a power failure.

Interface Capabilities

The analyzer has the following interface capabilities, as defined by the IEEE 488.1 standard:

SH1	full Source handshake capability
AH1	full Acceptor handshake capability
T6	basic Talker, Serial Poll, no Talk Only, unaddress if MLA
TE0	no Extended Talker capability
L4	basic Listener, no Listen Only, unaddress if MTA
LE0	no Extended Listener capability
SR1	full Service Request capability
RL1	full Remote/Local capability
DC1	full Device Clear capability
C1	System Controller capability
C2	send IFC and take charge Controller capability
C3	send REN Controller capability
C4[1]	respond to SRQ
C8[1]	send IFC, receive control, pass control, pass control to self
C12[2]	send IF messages, receive control, pass control
E2	tri-state drivers
DT1	full device trigger capability
PP0	no parallel poll capability

1 only when an HP Instrument BASIC program is running

2 only when an HP Instrument BASIC program is not running

Programming Fundamentals

This section includes specific information for programming your network analyzer. It includes how the analyzer interacts with a controller, how data is transferred between the analyzer and a controller, and how to use the analyzer's status register structure to generate service requests.

Controller Capabilities

The analyzer can be configured as an HP-IB system controller or as a talker/listener on the bus. To configure the analyzer, select either the `System Controller` or the `Talker/Listener` softkey in the [SYSTEM OPTIONS] `HP-IB` menu.

The analyzer is not usually configured as the system controller unless it is the only controller on the bus. This setup would be used if the analyzer only needed to control printers or plotters. It would also be used if HP Instrument BASIC was being used to control other test equipment.

When the analyzer is used with another controller on the bus, it is usually configured as a talker/listener. In this configuration, when the analyzer is passed control it can function as the active controller.

Response to Bus Management Commands

The HP-IB contains an attention (ATN) line that determines whether the interface is in command mode or data mode. When the interface is in command mode (ATN TRUE) a controller can send bus management commands over the bus. Bus management commands specify which devices on the interface can talk (send data) and which can listen (receive data). They also instruct devices on the bus, either individually or collectively, to perform a particular interface operation.

This section describes how the analyzer responds to the HP-IB bus management commands. 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.

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 `*OPC` command or query.

The command does not affect:

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

Go To Local (GTL)

This command returns the analyzer to local (front-panel) control. All keys on the analyzer's front-panel are enabled.

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 designated as 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.

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 remote softkey menu (including the `Return to LOCAL` softkey) 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, enabling all other front-panel keys. When the key is disabled, it does not allow the front-panel operator to return the analyzer to local mode.

Parallel Poll

The analyzer ignores all of the following parallel poll commands:

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

Remote Enable (REN)

REN is a single line on the HP-IB. When it is set TRUE, the analyzer will enter the remote mode when addressed to listen. It will remain 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 softkeys. The remote softkey menu includes seven keys that are available for use by a program. The eighth softkey is the `Return to LOCAL` key which allows a front-panel operator to return the analyzer to local mode, enabling all other front-panel keys.

Selected Device Clear (SDC)

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

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 *OPC command or query.

The command does not affect:

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

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 acts as the Request Service (RQS) bit. If the bit is set, 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.

Take Control Talker (TCT)

If 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.

Message Exchange

The analyzer communicates with the controller and other devices on the HP-IB using 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 discussed in Chapter 10.

There are two important things to remember about the message exchanges between the analyzer and other devices on the bus:

- The analyzer only talks after it receives a terminated query (see "Query Response Generation" later in this section).
- 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.

Input Queue.

The input queue temporarily stores the following until they are read by the analyzer's command parser:

- Device commands and queries.
- The HP-IB END message (EOI asserted while the last data byte is on the bus).

The input queue also 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 holds up to 128 bytes. It is cleared when:

- The analyzer is turned on.
- The Device Clear (DCL) or Selected Device Clear (SDC) command is received.

Error Queue.

The error queue temporarily stores up to 20 error messages. Each time the analyzer detects an error, it places a message in the queue. When you send the SYST:ERR? 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:

- All the error messages are read using the SYST:ERR? query.
- The analyzer is turned on.
- The *CLS command is received.

Output Queue.

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

- The message is read by a controller.
- The analyzer is turned on.
- The Device Clear (DCL) or Selected Device Clear (SDC) command is received.

Command Parser

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

One of the parser's most important functions is to determine the position of a program message in the analyzer's command tree (described in Chapter 10). When the command parser is reset, the next command it receives is expected to arise from the base of the analyzer's command tree.

The parser is reset when:

- The analyzer is turned on.
- The Device Clear (DCL) or Selected Device Clear (SDC) command is received.
- A colon immediately follows a semicolon in a program message. (For more information see "Sending Multiple Commands" in Chapter 10.)
- A program message terminator is received. A program message terminator can be an ASCII carriage return ($^C{}_R$) or newline character or the HP-IB END message (EOI set true).

Query Response Generation

When the analyzer parses a query, the response to that query is placed in the analyzer's output queue. The response should be read immediately after the query is sent. This ensures that the response is not cleared before it is read. The response is cleared when one of the following message exchange conditions occurs:

- Unterminated condition — the query is not properly terminated with an ASCII carriage return character or the HP-IB END message (EOI set true) before the response is read.
- Interrupted condition — a second program message is sent before the response to the first is read.
- Buffer deadlock — a program message is sent that exceeds the length of the input queue or that generates more response data than fits in the output queue.

2

Synchronizing the Analyzer and a Controller

Synchronizing the Analyzer and a Controller

The IEEE 488.2 standard provides tools that can be used to synchronize the analyzer and a controller. Proper use of these tools ensures that the analyzer is in a known state when you send a particular command or query.

Device commands can be divided into two broad classes:

- Sequential commands.
- Overlapped commands.

Most of the analyzer's commands are processed sequentially. A sequential command holds off the processing of subsequent commands until it has been completely processed.

Some commands do not hold off the processing of subsequent commands; they are called overlapped commands.

Overlapped Commands

Typically, overlapped commands take longer to process than sequential commands. For example, the `:INITIATE:IMMEDIATE` command restarts a measurement. The command is not considered to have been completely processed until the measurement is complete. This can take a long time with a narrow or fine system bandwidth or when averaging is enabled.

The analyzer has the following overlapped commands:

```
ABORt
CALibration:ZERO:AUTO
CONFigure[1|2]
DIAGnostic:CCONstants:LOAD
DIAGnostic:CCONstants:STORe:DISK
DIAGnostic:CCONstants:STORe:EEPRom
DIAGnostic:DITHer
DIAGnostic:SPUR:AVOid
HCOPy[:IMMediate]
INITiate[1|2]:CONTinuous
INITiate[1|2][:IMMediate]
MMEMory:LOAD:STATe
OUTPut[:STATe]
POWer[1|2]:MODE
PROGram[:SELected]:EXECute
SENSe[1|2]:AVERage:CLEar
SENSe[1|2]:AVERage:COUNt
SENSe[1|2]:AVERage[:STATe]
SENSe[1|2]:BWIDth[:RESolution]
SENSe[1|2]:CORRection:COLLect[:ACQuire]
SENSe[1|2]:CORRection:COLLect:ISTate[:AUTO]
SENSe[1|2]:CORRection:COLLect:METHod
SENSe[1|2]:CORRection:COLLect:SAVE
SENSe[1|2]:CORRection:CSET[:SELect]
SENSe[1|2]:CORRection[:STATe]
SENSe:COUPle
SENSe[1|2]:DETector[:FUNCtion]
SENSe[1|2]:DISTance:STARt (Option 100 only)
SENSe[1|2]:DISTance:STOP (Option 100 only)
SENSe[1|2]:FREQuency:CENTer
```

```
SENSe[1|2]:FREQuency:MODE (Option 100 only)
SENSe[1|2]:FREQuency:SPAN
SENSe[1|2]:FREQuency:SPAN:MAXimum
SENSe[1|2]:FREQuency:STARt
SENSe[1|2]:FREQuency:STOP
SENSe[1|2]:FUNCtion
SENSe[1|2]:FUNCtion:SRL:SCAN[:IMMediate] (Option 100 only)
SENSe:ROSCillator:SOURce
SENSe[1|2]:STATe
SENSe[1|2]:SWEep:POINts
SENSe[1|2]:SWEep:TIME
SENSe[1|2]:SWEep:TIME:AUTO
SENSe:SWEep:TRIGger:SOURce
SOURce[1|2]:POWer[:LEVel][:IMMediate][:AMPLitude]
SYSTem:PRESet
TRACe[:DATA]
TRIGger[:SEQuence]:SOURce
```

The NPO Flag

The analyzer uses a No Pending Operation (NPO) flag to keep track of overlapped commands. The NPO flag is reset to 0 when an overlapped command has not completed (still pending). It is set to 1 when no overlapped commands are pending. The NPO flag cannot be read directly but all of the following common commands take some action based on the setting of the flag.

`*WAI` Holds off the processing of subsequent commands until the NPO flag is set to 1. This ensures that commands in the analyzer's input queue are processed in the order received.

The program continues to run, and additional commands are received and parsed by the analyzer (but not executed), while waiting for the NPO flag to be set. Use of the `*WAI` command is explained later in this section and is demonstrated in the `SETUP` example program.

`*OPC?` Places a 1 in the analyzer's output queue when the NPO flag is set to 1. If the program is designed to read the output queue before it continues, this effectively pauses the controller until all pending overlapped commands are completed. Use of the `*OPC?` command is explained later in this chapter and is demonstrated in the `TRANCAL` and `REFLCAL` example programs.

`*OPC` Sets bit 0 of the Standard Event Status event register to 1 when the NPO flag is set to 1. The analyzer's status registers can then be used to generate a service request when all pending overlapped commands are completed. This synchronizes the controller to the completion of an overlapped command, but also leaves the controller free to perform other tasks while the command is executing.

NOTE

`*OPC` only informs you when the NPO flag is set to 1. It does not hold off the processing of subsequent commands. No commands should be sent to the analyzer between sending the `*OPC` command and receiving the service request. Any command sent will be executed and may affect how the instrument responds to the previously sent `*OPC`.

The *CLS and *RST commands cancel any preceding *OPC command or query. Pending overlapped commands are still completed, but their completion is not reported in either the status register or the output queue. Two HP-IB bus management commands — Device Clear (DCL) and Selected Device Clear (SDC) — also cancel any preceding *OPC command or query.

NOTE

Use *WAI, *OPC? or *OPC whenever overlapped commands are used. A recommended technique is to send *OPC? at the end of each group of commands.

CAUTION

ALWAYS trigger an individual sweep (using *OPC? and waiting for the reply) before reading data over the bus or executing a marker function. The analyzer has the ability to process the commands it receives faster than it can make a measurement. If the measurement is not complete when the data is read or a marker search function is executed the results are invalid.

The command to use (in an IBASIC OUTPUT statement) is:

```
OUTPUT @Hp8711;"ABOR;:INIT:CONT OFF;:INIT;*OPC?"
ENTER @Hp8711;Opc_done
```

or another form of the :INITiate[1|2][:IMMediate] command combined with the *OPC? query.

Refer to "Taking Sweeps" in Chapter 6 for more information.

Usage of *WAI and *OPC?

***WAI**

The following example describes the use of the *WAI command. For this discussion, remember that a sequential command holds off the processing of subsequent commands until it has been completely processed. An overlapped command does not.

```
10 OUTPUT @Rfna;"command1"
20 OUTPUT @Rfna;"command2;*WAI"
30 OUTPUT @Rfna;"command3;"
40 OUTPUT @Rfna;"command4"
50 END
```

In the example above:

- Commands 1 through 4 are sent to the analyzer as fast as the HP-IB bus traffic will allow, and the program may very well end before any command has been completed.
- Command 1 begins execution first.
- The order in which commands 1 and 2 are *completed* depends on the command types. If both commands are overlapped commands (versus sequential commands), the order of completion is unknown.
- Commands 3 and 4 will not be parsed until commands 1 and 2 are completed.
- Command 3 will begin execution before command 4.
- The order in which commands 3 and 4 are *completed* depends on the command types. If both commands are overlapped commands (versus sequential commands), the order of completion is unknown.

***OPC?**

The following example describes the use of the *OPC? query and command. For this discussion, remember that a sequential command holds off the processing of subsequent commands until it has been completely processed. An overlapped command does not.

```
10 OUTPUT @Rfna;"command1"
20 OUTPUT @Rfna;"command2;*OPC?"
30 ENTER @Rfna;OPC
40 OUTPUT @Rfna;"command3;"
50 OUTPUT @Rfna;"command4;*OPC?"
60 ENTER @Rfna;OPC
70 END
```

In the example above:

- Commands 1 and 2 are sent to the analyzer as fast as the HP-IB bus traffic will allow.
- Command 1 will begin execution before command 2.
- The order in which commands 1 and 2 are *completed* depends on the command types. If both commands are overlapped commands (versus sequential commands), the order of completion is unknown.
- When commands 1 and 2 are completed, commands 3 and 4 will be sent to the analyzer as fast as the HP-IB bus traffic will allow.
- Command 3 will begin execution before command 4.
- The order in which commands 3 and 4 are *completed* depends on the command types. If both commands are overlapped commands (versus sequential commands), the order of completion is unknown.
- This program will not end until the OPC in line 60 is returned.

3

Passing Control

Passing Control

When an external controller is connected to the analyzer with an HP-IB cable, passing control may be needed to control devices such as printers and plotters that are also connected on the HP-IB. For some operations the active controller must pass control to the analyzer. When the analyzer completes the operation, it automatically passes control of the bus back to the external controller.

An example program, `PASSCTRL`, demonstrates passing control to the analyzer. In this example program control is passed so the analyzer can control a printer for hardcopy output. See Chapter 8, "Example Programs."

> **NOTE**
>
> Pass Control is not needed to control peripherals connected to the serial, parallel, or LAN ports.

For smooth passing of control, take steps that ensure the following conditions are met:

- The analyzer must know the controller's address so it can pass control back.
- The controller must be informed when the analyzer passes control back.

The following is a procedure for passing control:

1. Send the controller's HP-IB address to the analyzer with the *PCB command.
2. Clear the analyzer's status registers with the *CLS command.
3. Enable the analyzer's status registers to generate a service request when the Operation Complete bit is set. (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 of the bus followed by the *OPC command.
6. Pass control to the analyzer and wait for the service request. The service request indicates that the command has been completed and control has been passed back to the controller.

NOTE

For this procedure to work properly, only the command that requires control of the bus should be pending. Other overlapped commands should not. For more information on overlapped commands, see Chapter 2, "Synchronizing the Analyzer and a Controller."

4

Data Types and Encoding

Data Types and Encoding

Data is transferred between the analyzer and a controller via the HP-IB data lines, DIO1 through DIO8. Such transfers occur in a byte-serial (one byte at a time), bit-parallel (8 bits at a time) manner. This section discusses the following aspects of data transfer:

- The different data types used during data transfers.
- Data encoding used during transfers of numeric block data.

Data Types

The analyzer uses a number of different data types during data transfers. Data transfer occurs in response to a query. The data type used is determined by the parameter being queried. The different parameter types are described in the "Parameter Types" section of Chapter 10. Data types described in this section are:

- Numeric Data.
- Character Data
- String Data
- Expression Data
- Block Data

Numeric Data

The analyzer returns three types of numeric data in response to queries:

NR1 data	Integers (such as `+1`, `0`, `-1`, `123`, `-12345`). This is the response type for boolean parameters as well as some numeric parameters.
NR2 data	Floating point numbers with an explicit decimal point (such as `12.3`, `+1.234`, `-0.12345`).
NR3 data	Floating point numbers in scientific notation (such as `+1.23E+5`, `+123.4E-3`, `-456.789E+6`).

Character Data

Character data consists of ASCII characters grouped together in mnemonics that represent specific instrument settings (such as `MAXimum` , `MINimum` or `MLOGarithmic`). The analyzer always returns the short form of the mnemonic in upper-case alpha characters.

String Data

String data consists of ASCII characters. The string must be enclosed by a delimiter, either single quotes (`'This is string data.'`) or double quotes (`"This is also string data."`). To include the delimiter as a character in the string it must be typed twice without any characters in between. The analyzer always uses double quotes when it returns string data.

Expression Data

Expression data consists of mathematical expressions that use character parameters. When expression data is sent to the analyzer it is always enclosed in parentheses (such as `(IMPL/CH1SMEM)` or `(IMPL)`). The analyzer returns expression data enclosed in double quotes.

Block Data

Block data are typically used to transfer large quantities of related data (like a data trace). Blocks can be sent as definite length blocks or indefinite length blocks — the instrument will accept either form. The analyzer always returns definite length block data in response to queries.

Definite Block Length

The general form for a definite block length transfer is:

```
#<num_digits><num_bytes><data_bytes>
```

In the definite length block, two numbers must be specified. The single decimal digit **<num_digits>** specifies how many digits are contained in **<num_bytes>**. The decimal number **<num_bytes>** specifies how many data bytes will follow in **<data_bytes>**. An example IBASIC (or HP BASIC) statement to send ABC+XYZ as a definite block length parameter is shown, note that the data block contains seven bytes (7) and only one digit is needed to describe the block length 1.

```
OUTPUT 716;"#17ABC+XYZ"
```

NOTE

This analyzer will send an additional <CR> with EOI asserted for definite block length transfers. The definite length block form for your analyzer is:

```
#<num_digits><num_bytes><data_bytes><CR><EOI>
```

<num_bytes> is the number of **<data_bytes>** without counting **<CR><EOI>**.

Indefinite Block Length

The general form for an indefinite block length transfer is:

```
#0<data_bytes><CR><EOI>
```

After the last data byte is sent, the indefinite length block must be terminated by sending a carriage return or newline with EOI asserted. This forces the termination of the program message. An example IBASIC (or HP BASIC) statement to send ABC+XYZ as an indefinite block length parameter is shown, note that ,END is used to properly terminate the message.

```
OUTPUT 716;"#0ABC+XYZ",END
```

Data Encoding for Large Data Transfers

The `FORMat:DATA` command selects the type of data and the type of data encoding that is used to transfer large blocks of numeric data between the analyzer and a controller. There are two specifiers:

`REAL` specifies the block data type. Either the definite or indefinite length syntax can be used. The block is transferred as a series of binary-encoded floating-point numbers. Data transfers of the `REAL,64` data type are demonstrated in the `REALDATA` example program.

`INTeger` specifies the block data type. Either the definite or indefinite length syntax can be used. The block is transferred as an array of binary-encoded data with each point represented by a set of three 16-bit integers. This is the instrument's internal format — it should only be used for data that will be returned to the instrument for later use. Data transfers of the `INTeger,16` data type are demonstrated in the `INTDATA` and `LOADCALS` example programs.

`ASCii` specifies the numeric data type (NR1, NR2 or NR3 syntax). The data is transferred as a series of ASCII-encoded numbers separated by commas. `ASCii` formatted data transfers are demonstrated in the `ASCDATA` example program.

Blocks that contain mixed data — both numbers and ASCII characters — ignore the setting of `FORMat:DATA`. These blocks always transfer as either definite length or indefinite length block data. The following commands transfer blocks of mixed data:

```
PROGram[:SELected]:DEFine
SYSTem:SET
```

ASCII Encoding

The ANSI X3.4-1977 standard defines the ASCII 7-bit code. When an ASCII-encoded byte is sent over the HP-IB, bits 0 through 6 of the byte (bit 0 being the least significant bit) correspond to the HP-IB data lines DIO1 through DIO7. DIO8 is ignored.

When ASCII encoding is used for large blocks of data, the number of significant digits to be returned for each number in the block can be specified. For example, the following command returns all numbers as NR3 data with 7 significant digits.

```
FORMat:DATA ASCii,7
```

Binary Encoding

When binary encoding is used for large blocks of data, all numbers in the block are transferred as 32-bit or 64-bit binary floating point numbers or as an array of 16-bit integers. The binary floating-point formats are defined in the IEEE 754-1985 standard.

`FORMat:DATA REAL,32`	*selects the IEEE 32-bit format (not supported by IBASIC or HP BASIC).*
`FORMat:DATA REAL,64`	*selects the IEEE 64-bit format.*
`FORMat:DATA INTeger,16`	*selects the 16-bit integer format.*

Byte Swapping

PC compatibles frequently use a modification of the IEEE floating point formats with the byte order reversed. To reverse the byte order for data transfer into a PC, the `FORMat:BORDer` command should be used.

`FORMat:BORDer SWAPped`	*selects the byte-swapped format*
`FORMat:BORDer NORMal`	*selects the standard format*

5

Using Status Registers

Using Status Registers

The analyzer's status registers contain information about the condition of the network analyzer and its measurements. This section describes the registers and their use in HP-IB programming.

Example programs using the status registers are included in Chapter 8, "Example Programs." These programs include `SRQ` and `GRAPHICS` which use service request interrupt routines, `PASSCTRL` which uses the status byte to request control of the HP-IB and `LIMITEST` which uses the Limit Fail condition register.

General Status Register Model

The analyzer's status system is based on the general status register model shown in Figure 5-1. Most of the analyzer's register sets include all of the registers shown in the model, although commands are not always available for reading or writing a particular register. The information flow within a register set starts at the condition register and ends at the register summary bit (see Figure 5-2). This flow is controlled by setting bits in the transition and enable registers.

Two register sets — the Status Byte and the Standard Event Status Register — are 8-bits wide. All others are 16-bits wide, but the most significant bit (bit 15) in the larger registers is always set to 0.

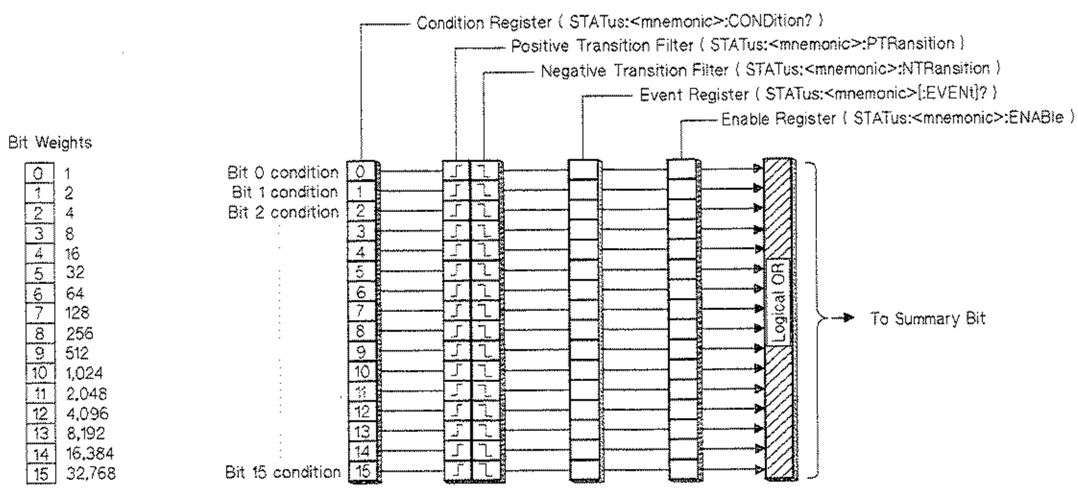


Figure 5-1. General Status Register Model

Condition Register

Condition registers continuously monitor the instrument's hardware and firmware status. Bits in a condition register are not latched or buffered, they are updated in real time. When the condition monitored by a specific 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.

Transition Registers

Transition registers control what type of change in a condition register will set the corresponding bit in the event register. Positive state transitions (0 to 1) are only reported to the event register if the corresponding positive transition bit is set to 1. Negative state transitions (1 to 0) are only reported if the corresponding negative transition bit is set to 1. Setting both transition bits to 1 causes both positive and negative changes to be reported. Transition registers are read-write, and are unaffected by `*CLS` (clear status) or queries. They are reset to instrument default conditions at power up and after `*RST` and `SYSTem:PRESet` commands.

Event Register

Event registers latch any reported 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. It remains set until the event register is cleared. Event registers are read-only.

An event register is cleared when you read it. All event registers are cleared when you send the `*CLS` (clear status) command.

Enable Register

Enable registers control the reporting of events (latched conditions) to the register summary bit. If an enable bit is set to 1 the corresponding event is included in the logical ORing process that determines the state of the summary bit. (The summary bit is only set to 1 if one or more enabled event bits are set to 1.) Summary bits are recorded in the instrument's status byte. Enable registers are read-write and are cleared by *CLS (clear status).

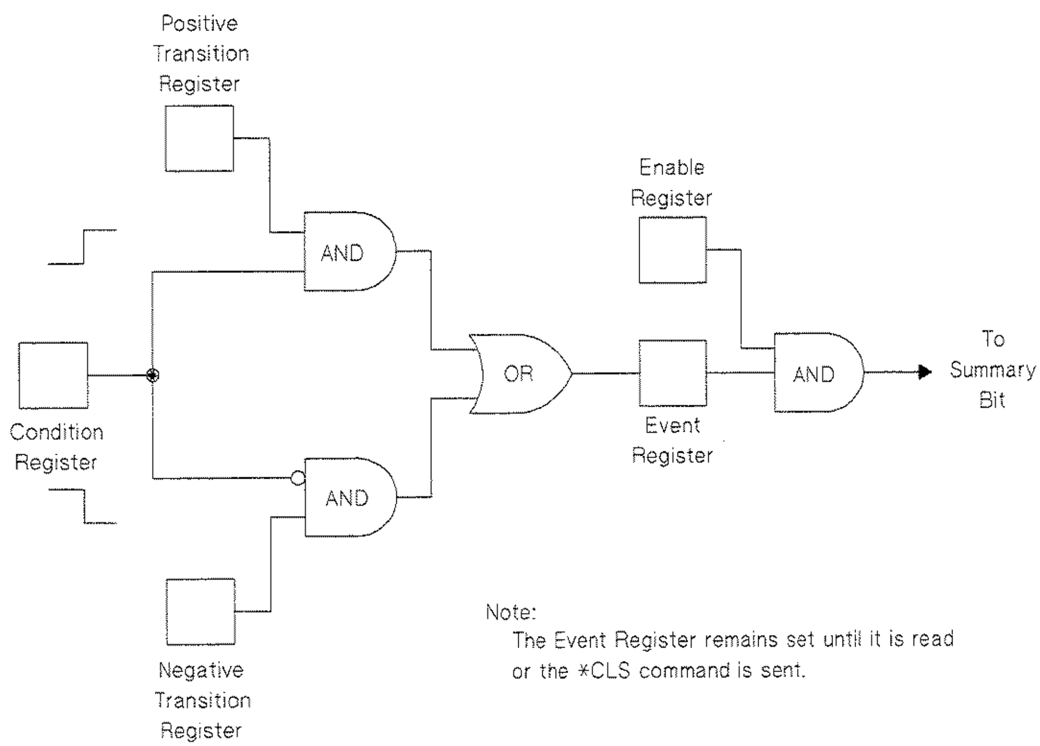


Figure 5-2. Flow of information within a register set

How to Use Registers

There are two methods of accessing the information in status registers:

- The direct-read method.
- The service request (SRQ) method.

In the direct-read method the analyzer is passive. It only tells the controller that conditions have changed when the controller asks the right question. In the SRQ method, the analyzer is more active. It tells the controller when there has been a condition change without the controller asking. Either method allows you to monitor one or more conditions.

The following steps are used to monitor a condition with the direct read method:

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 direct-read method works well when it is not necessary to know about changes the moment they occur. It does not work well if immediate knowledge of the condition change is needed. A program that used this method to detect a change in a condition would need to continuously read the registers at very short intervals. The SRQ method is better suited for that type of need.

The Service Request Process

The following steps are used to monitor a condition with the SRQ method:

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. The controller's response to the SRQ is determined by the program being run.

Generating a Service Request

A service request is generated using the Status Byte. As shown in Figure 5-3, the analyzer's other register sets report to the Status Byte. Some of them report directly while others report indirectly through other register sets.

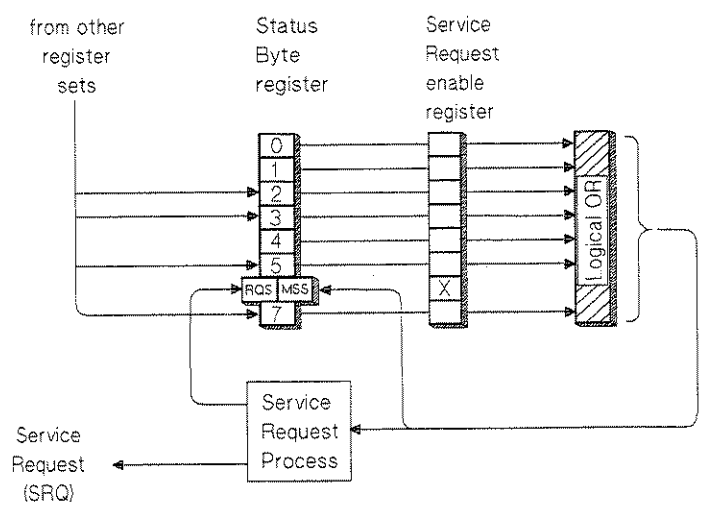


Figure 5-3. Generating a Service Request

The process of preparing the analyzer to generate a service request, and the handling of that interrupt when it is received by a program, are demonstrated in the SRQ example program.

When a register set causes its summary bit in the Status Byte to change from 0 to 1, the analyzer can initiate the service request (SRQ) process. If both the following conditions are true the process is initiated:

- 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 and sets the Status Byte's request service (RQS) bit to 1. Both actions are necessary to inform the controller that the analyzer requires service. Setting the SRQ line informs the controller that some device on the bus requires service. Setting the RQS bit allows the controller to determine that the analyzer was the device that initiated the request.

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

As implied in Figure 5-3, bit 6 of the Status Byte register serves two functions; the request service function (RQS) and the master summary status function (MSS). Two different methods for reading the register allow you to access the two functions. Reading the register with a serial poll allows you to access the bit's RQS function. Reading the register with *STB allows you to access the bit's MSS function.

The Analyzer's Status Register Sets

The analyzer uses eight register sets to keep track of instrument status:

Status Byte	*STB? and *SRE
Device Status	STATus:DEVice
Limit Fail	STATus:QUEStionable:LIMit
Questionable Status	STATus:QUEStionable
Standard Event Status	*ESR? and *ESE
Measuring Status	STATus:OPERation:MEASuring
Averaging Status	STATus:OPERation:AVERaging
Operational Status	STATus:OPERation

Their reporting structure is summarized in Figure 5-4. They are described in greater detail in the following section.

NOTE

Register bits not explicitly presented in the following sections are not used by the analyzer. A query to one of these bits returns a value of 0.

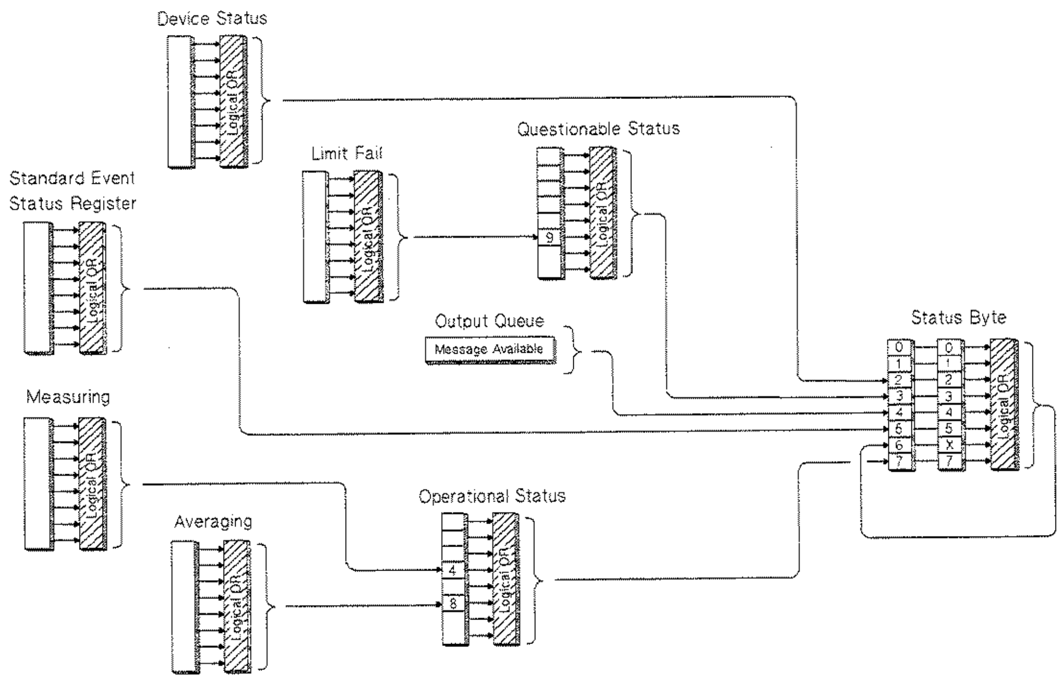


Figure 5-4. Analyzer Register Sets

Status Byte

The Status Byte register set summarizes the states of the other register sets and monitors the analyzer's output queue. It is also responsible for generating service requests (see "Generating a Service Request" earlier in this chapter). See Figure 5-5.

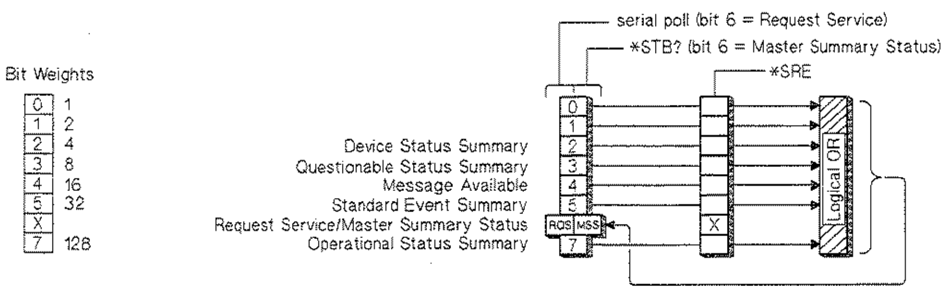


Figure 5-5. The Status Byte Register Set

The Status Byte register set does not conform to the general status register model described at the beginning of this chapter. It contains only two registers: the Status Byte register and the Service Request enable register. The Status Byte register behaves like a condition register for all bits except bit 6. The Service Request enable register behaves like a standard enable register except that bit 6 is always set to 0.

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

Device Status Summary	(bit 2) is set to 1 when one or more enabled bits in the Device Status event register are set to 1.
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 Status Summary	(bit 5) is set to 1 when one or more enabled bits in the Standard Event Status 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).
Operational Status Summary	(bit 7) is set to 1 when one or more enabled bits in the Operational Status event register are set to 1.

The commands used to read and write the Status Byte registers are listed below:

`SPOLL`	an IBASIC (or HP BASIC) command used in the service request process to determine which device on the bus is requesting service.
`*STB?`	reads the value of the instrument's status byte. This is a non-destructive read, the Status Byte is cleared by the `*CLS` command.
`*SRE <num>`	sets bits in the Service Request Enable register. The current setting of the Service Request Enable register is stored in non-volatile memory. If `*PSC` has been set, it will be saved at power on.
`*SRE?`	reads the current state of the Service Request Enable register.

Device Status Register Set

The Device Status register set monitors the state of device-specific parameters.

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

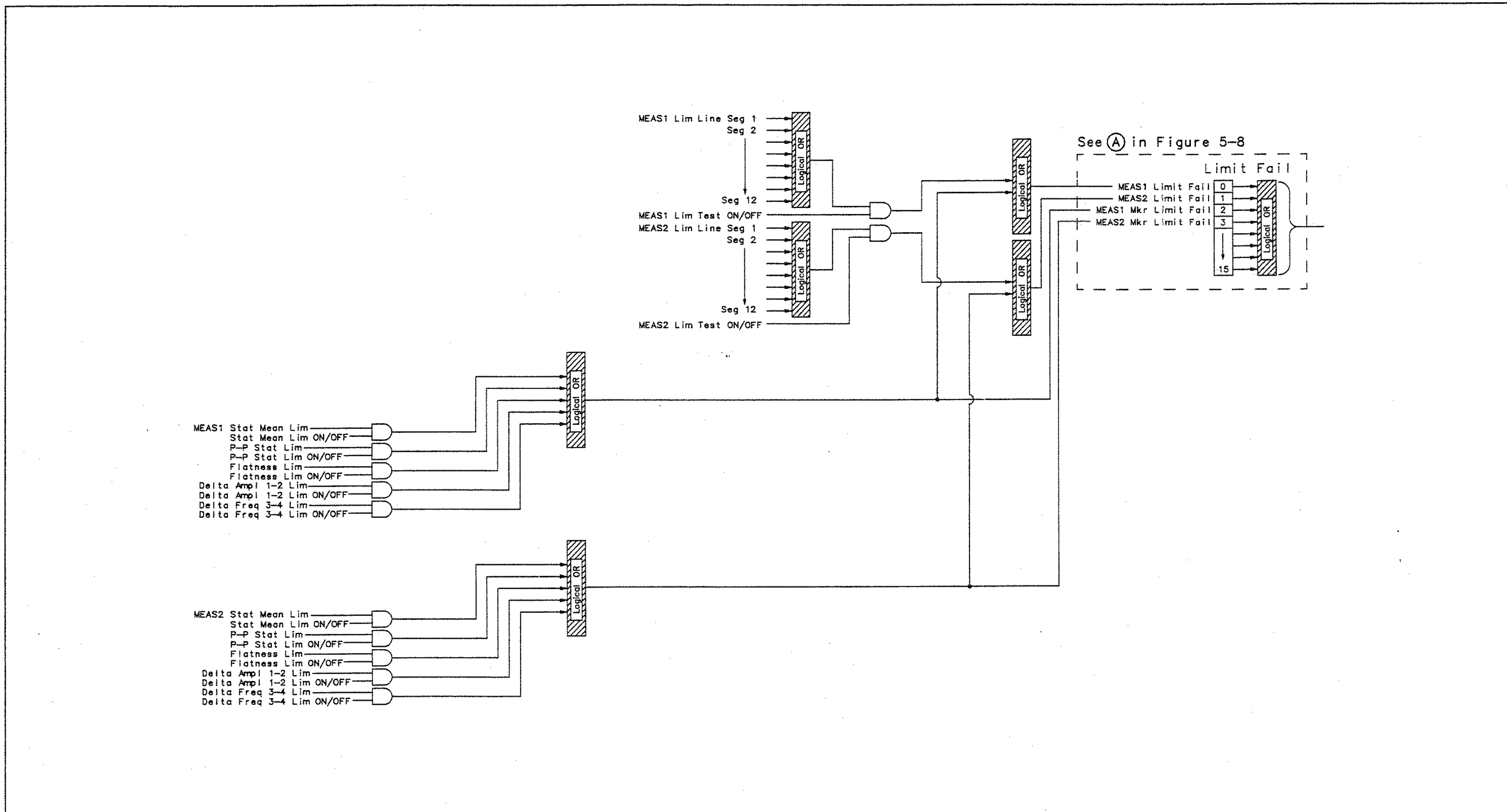
Key Pressed	(bit 0) is set to 1 when one of the analyzer's front panel keys has been pressed.
Any Softkey Pressed	(bit 1) is set to 1 when one of the analyzer's softkeys has been pressed.
Any External Keyboard Key Pressed	(bit 2) is set to 1 when a key has been pressed on an external keyboard connected to the DIN KEYBOARD connector on the rear panel of the analyzer.
Front Panel Knob Turned	(bit 3) is set to 1 when the analyzer's front panel knob is turned.

Limit Fail Register Set

The Limit Fail register set monitors limit test results for both measurement channels.

Bits in the Limit Fail condition register are set to 1 under the following conditions (refer also to Figure 5-6.)

Measurement Channel 1 Limit Failed	(bit 0) is set to 1 when limit testing is enabled and any point on measurement channel 1 fails the limit test, *or* when any enabled marker limit on measurement channel 1 has failed.
Measurement Channel 2 Limit Failed	(bit 1) is set to 1 when limit testing is enabled and any point on measurement channel 2 fails the limit test, *or* when any enabled marker limit on measurement channel 2 has failed.
Measurement Channel 1 Marker Limit Failed	(bit 2) is set to 1 when any enabled marker limit on measurement channel 1 has failed.
Measurement Channel 2 Marker Limit Failed	(bit 3) is set to 1 when any enabled marker limit on measurement channel 2 has failed.


MEAS1 Lim Line Seg 1
Seg 2
Seg 12
MEAS1 Lim Test ON/OFF
MEAS2 Lim Line Seg 1
Seg 2
Seg 12
MEAS2 Lim Test ON/OFF
Logical OR
See A in Figure 5-8
Limit Fail
MEAS1 Limit Fail
MEAS2 Limit Fail
MEAS1 Mkr Limit Fail
MEAS2 Mkr Limit Fail
0
1
2
3
15
MEAS1 Stat Mean Lim
Stat Mean Lim ON/OFF
P-P Stat Lim
P-P Stat Lim ON/OFF
Flatness Lim
Flatness Lim ON/OFF
Delta Ampl 1-2 Lim
Delta Ampl 1-2 Lim ON/OFF
Delta Freq 3-4 Lim
Delta Freq 3-4 Lim ON/OFF
MEAS2 Stat Mean Lim
Stat Mean Lim ON/OFF
P-P Stat Lim
P-P Stat Lim ON/OFF
Flatness Lim
Flatness Lim ON/OFF
Delta Ampl 1-2 Lim
Delta Ampl 1-2 Lim ON/OFF
Delta Freq 3-4 Lim
Delta Freq 3-4 Lim ON/OFF

cd61c

Fig. 5-6

Figure LIMFAIL here.

Figure 5-6. The Limit Fail Register Set

Insert 9x17 drawing cd61c

Questionable Status Register Set

The Questionable Status register set monitors conditions that affect the quality of measurement data.

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

Limit Fail (bit 9) is set to 1 when one or more enabled bits in the Limit Fail event register are set to 1.

Data Questionable (bit 10) is set to 1 when a change in the analyzer's configuration requires that new measurement data be taken.

Standard Event Status Register Set

The Standard Event Status register set monitors HP-IB errors and synchronization conditions. See Figure 5-7.

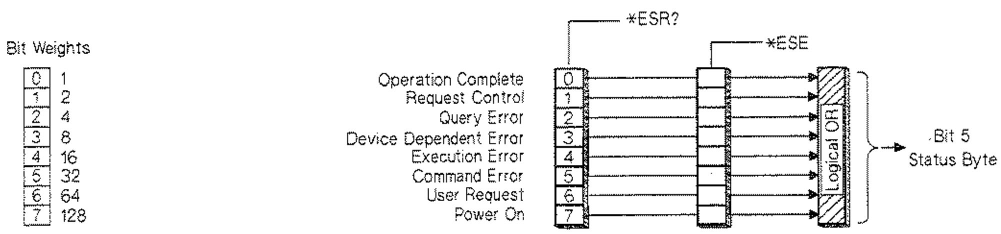


Figure 5-7. The Standard Event Status Register Set

The Standard Event Status register set does not conform to the general status register model described at the beginning of this section. It contains only two registers: the Standard Event Status event register and the Standard Event Status enable register. The Standard Event Status event register is similar to other event registers, but behaves like a register set 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.

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

- The *OPC command is sent to the analyzer.
- The analyzer completes all pending overlapped commands.

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

- The analyzer is configured as a talker/listener for HP-IB operation.
- 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 when the command parser detects a query error. A query error indicates:

1. an attempt to read data from the Output Queue when no data was present.
2. that data in the Output Queue was lost. An example of this would be queue overflow.

Device Dependent Error (bit 3) is set to 1 when the command parser detects a device-dependent error. A device-dependent error is any analyzer operation that did not execute properly due to some internal condition such as overrange. This bit indicates that the error was not a command, query, or an execution error.

Execution Error (bit 4) is set to 1 when the command parser detects an execution error. Execution errors occur when:

1. a <PROGRAM DATA> element received in a command was outside the legal range for the analyzer, or inconsistent with the operation of the analyzer.
2. the analyzer could not execute a valid command due to some analyzer condition.

Command Error (bit 5) is set to 1 when the command parser detects a command error. The following events cause a command error:

1. An IEEE 488.2 syntax error. This means that the analyzer received a message that did not follow the syntax defined by the 488.2 standard.
2. A semantic error occurred. For example, the analyzer received an incorrectly spelled command. Another

example would be that the analyzer received an optional 488.2 command that it does not implement.

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

The commands used to read and write the Standard Event Status registers are listed below:

`*ESR?` reads the value of the standard event status register.

`*ESE <num>` sets bits in the standard event status enable register. The current setting of the standard event statue enable register is stored in non-volatile memory. If `*PSC` has been set, it will be saved at power on.

`*ESE?` reads the current state of the standard event status enable register.

Measuring Status Register Set

The Measuring Status register set monitors conditions in the analyzer's measurement process.

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

Channel 1 Measuring (bit 0) is set to 1 while the analyzer is collecting measurement data on channel 1.

Channel 2 Measuring (bit 1) is set to 1 while the analyzer is collecting measurement data on channel 2.

Averaging Status Register Set

The Averaging Status register set monitors conditions in the analyzer's measurement process when the trace averaging function is in use.

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

Measurement Channel 1 Averaging (bit 0) is set to 1 while the analyzer is sweeping on measurement channel 1 and the number of sweeps completed (since "average restart") is less than the averaging factor.

Measurement Channel 2 Averaging (bit 1) is set to 1 while the analyzer is sweeping on measurement channel 2 and the number of sweeps completed (since "average restart") is less than the averaging factor.

Operational Status Register Set

The Operation Status register set monitors conditions in the analyzer's measurement process, disk operations, and printing/plotting operations. It also monitors the state of the current HP Instrument BASIC program.

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

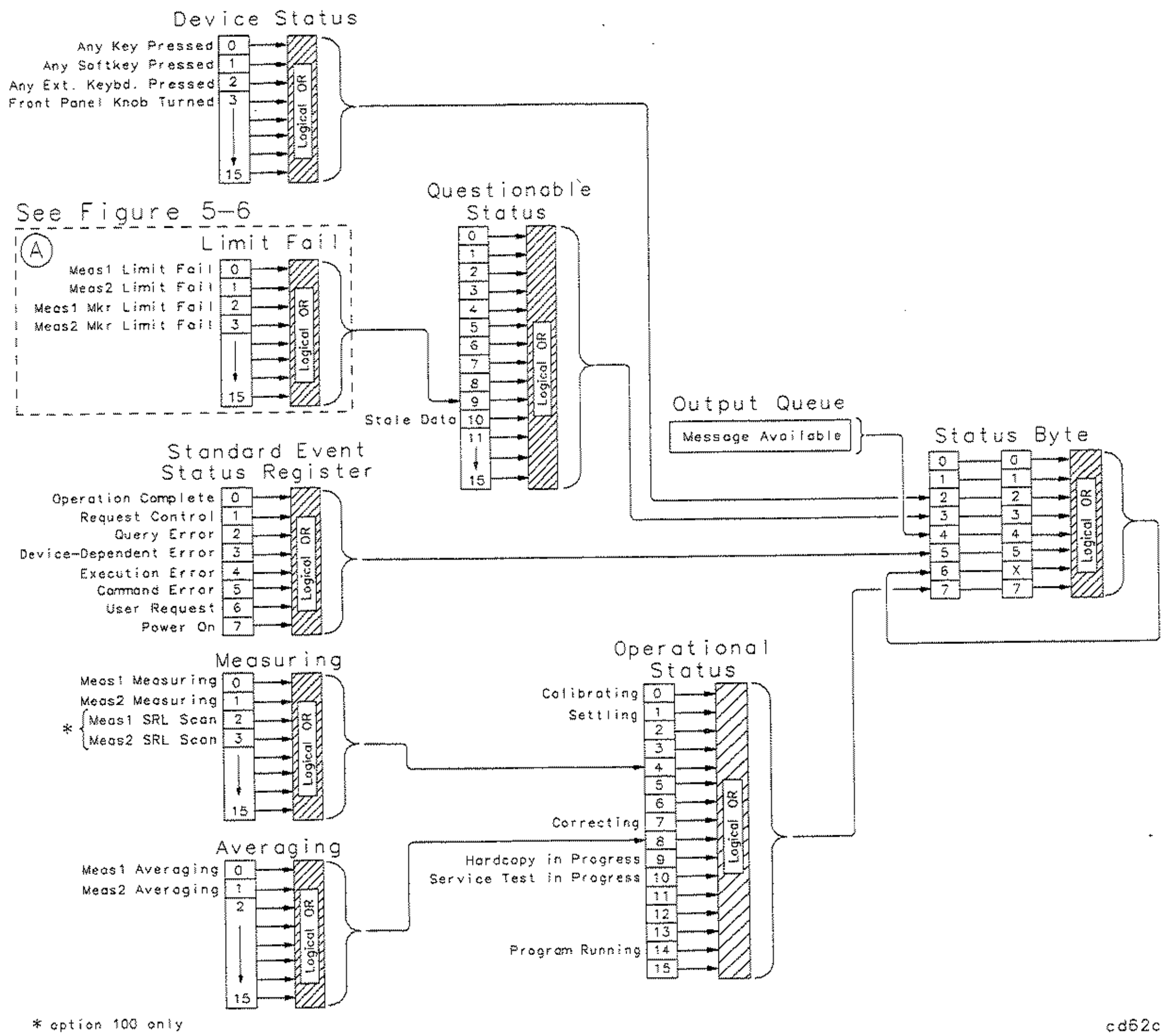
Calibrating	(bit 0) is set to 1 while the instrument is zeroing the broadband diode detectors.
Settling	(bit 1) is set to 1 while the measurement hardware is settling.
Measuring	(bit 4) is set to 1 when one or more enabled bits in the Measuring Status event register are set to 1.
Correcting	(bit 7) is set to 1 while the analyzer is performing a calibration function.
Averaging	(bit 8) is set to 1 when one or more enabled bits in the Averaging Status event register are set to 1.
Hardcopy Running	(bit 9) is set to 1 while the analyzer is performing a hardcopy (print or plot) function.
Test Running	(bit 10) is set to 1 when one of the analyzer's internal service tests is being run.
Program Running	(bit 14) is set to 1 while an HP Instrument BASIC program is running on the analyzer's internal controller.

STATus:PRESet Settings

Executing the `STATus:PRESet` command changes the settings in the enable (`ENAB`), positive transition (`PTR`) and negative transition (`NTR`) registers. The table below shows the settings after the command is executed.

Register Set	ENABle	PTRansition	NTRansition
STATus:DEVice	all 0s	all 1s	all 0s
STATus:QUEStionable:LIMit	all 1s	all 1s	all 0s
STATus:QUEStionable	all 0s	all 1s	all 0s
STATus:OPERation:MEASuring	all 1s	all 0s	all 1s
STATus:OPERation:AVERaging	all 1s	all 0s	all 1s
STATus:OPERation	all 0s	all 1s	all 0s

Analyzer Register Set Summary



6

Trace Data Transfers

Trace Data Transfers

This chapter explains how to read (query) the measurement data trace from the analyzer into your program. It also describes how to send data from your program to the analyzer's measurement arrays. Accessing the measurement arrays is done using SCPI commands. If you are using IBASIC (Option 1C2), you can also access the measurement arrays using high-speed subroutines. Refer to the *HP Instrument BASIC User's Handbook* for more details.

Figure 6-1 is a data processing flow diagram that represents the flow of numerical data. The data passes through several math operations, denoted in the figure by single-line boxes. Most of these operations can be selected and controlled with the front panel CONFIGURE block menus. The data is stored in arrays along the way, denoted by double-line boxes. These arrays are places in the flow path where data is accessible via HP-IB. While only a single flow path is shown, two identical paths are available, corresponding to measurement channels 1 and 2.

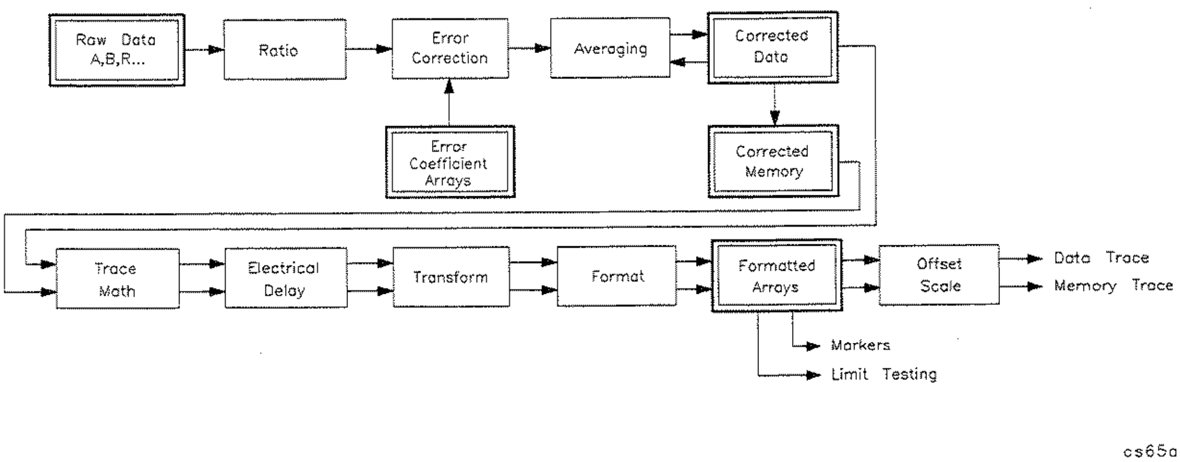


Figure 6-1. Numeric Data Flow Through the Network Analyzer

Querying the Measurement Trace Using BASIC

After making a measurement, you can read the resultant measurement trace out of the analyzer using the SCPI query

```
"TRACE:DATA?  CH1FDATA"
```

The BASIC program segment below shows how to read the trace from the analyzer into an array in your program.

```
10    REAL Trace(1:201)
20    ASSIGN @Hp8711 TO 716
30    ! Take sweep here
40    OUTPUT @Hp8711;"FORM:DATA ASCII,5"
50    OUTPUT @Hp8711;"TRACE:DATA? CH1FDATA"
60    ENTER @Hp8711;Trace(*)
70    DISP Trace(1),Trace(2),Trace(3),".  .  .  ."
```

In this program, the `TRACE:DATA?` query returns all of the measurement points as a single block. The analyzer computes the value for each point using the measurement format selected by the [FORMAT] menu (CALC:FORM SCPI command), and returns a block of data called the formatted data array. The values of each point correspond to the values displayed on the screen, or those shown in the marker readouts. The frequency stimulus value (X-axis) of each point is not returned by the `TRACE:DATA?` query; only the measurement response (Y-axis) values are returned.

When transferring the block of trace data, you may select either binary or ASCII data encoding. This is explained in Chapter 4 in the section titled "Data Encoding for Large Data Transfers." Notice that the terms "encoding format" and "measurement format" are not the same. The encoding format determines how the numbers are represented as bytes, while the measurement format corresponds to the meaning of the value of the numbers.

The easiest way to transfer a measurement data trace is to use ASCII data encoding.

In the example above, the array Trace(1:201) contains 201 real (floating point) numbers. The SCPI command `"FORM:DATA ASCII,5"` specifies ASCII data encoding, with 5 significant digits. The command `"TRACE:DATA? CH1FDATA"` instructs the analyzer to send the measurement trace. The ENTER statement reads the measurement data sent by the analyzer into the Trace(1:201) array.

It is important to make sure that the Trace array declared in your program is the same size as the measurement trace on the analyzer, or an error will occur. The ENTER statement attempts to read data from the analyzer until it completely fills the Trace array, at which point it expects to receive a end-of-data terminator from the analyzer. To be safe, your program should use the "SENS:SWE:POIN" SCPI command to set the number of measurement data points to the desired value.

Refer to the example program ASCDATA in Chapter 8 for a complete example.

Smith Chart and Polar Formats

Each measurement point is represented by a single floating point number. This is the case for all of the analyzer's measurement formats except Smith Chart and Polar in the HP 8712C and 8714C. When Smith Chart or Polar format is selected, each point is represented by two numbers, the first one being the real portion and the second being the imaginary portion of the complex measurement value.

Below is a modified example program that will work when using Smith Chart or Polar formats.

```
10  REAL Trace(1:201,1:2)
20  ASSIGN @Hp8711 TO 716
30  ! Take sweep here
40  OUTPUT @Hp8711;"FORM:DATA ASCII,5"
50  OUTPUT @Hp8711;"TRACE:DATA? CH1FDATA"
60  ENTER @Hp8711;Trace(*)
70  DISP Trace(1,1),Trace(1,2),". . . .",Trace(201,1),Trace(201,2)
```

Querying the Measurement Trace Using SICL

This section includes a complete SICL C program that shows how to read the measurement trace from the analyzer.

```
/**************************************************************************
 * This program takes a sweep, reads the trace, and prints it.
 * It uses SICL (Standard Instrument Control Library) to talk
 * to the analyzer over HP-IB.
 *
 * On HP-UX, compile using:    cc -Aa -o query_trace  query_trace.c -lsicl
 **************************************************************************/


#include <sicl.h>           /* For iopen(), iprintf(), iscanf(), INST, ... */
#include <stdio.h>          /* For printf() */

int main(void)
{
    INST analyzer;              /* Handle used to talk to analyzer */
    float data_buf[1601];       /* measurement trace.  32-bit floats */
    int num_trace_bytes;
    int pt;

    num_trace_bytes = sizeof(data_buf);   /* Set to max allowable bytes */

    /* Open the network analyzer at address 16 */
    analyzer = iopen("hpib,16");

    /* Clear the bus */
    iclear(analyzer);

    /* Abort current sweep and put analyzer sweep in hold */
    iprintf(analyzer, "ABORT\n");
    iprintf(analyzer, "INIT:CONT OFF\n");

    /* Take one sweep, wait until done */
    iprintf(analyzer, "INIT1\n");
    iprintf(analyzer, "*OPC?\n");
```

```
    iscanf(analyzer, "%*s");

    /* Request the trace data in 32-bit floating point format */
    iprintf(analyzer, "FORM:BORD NORM\n");
    iprintf(analyzer, "FORM:DATA REAL,32\n");

    /* Query the trace, read into data_buf[]. */
    iprintf(analyzer, "TRAC? CH1FDATA\n");
    iscanf(analyzer, "%#b%*c", &num_trace_bytes, &data_buf[0]);

    /* Print the trace values. */
    for (pt = 0; pt < num_trace_bytes/sizeof(float); pt++) {
        printf("%4d     %g\n", pt, data_buf[pt]);
    }

    /* Close analyzer and exit. */
    iclose(analyzer);
    return 0;
}
```

Using Binary Data Encoding

The previous section describes how to query the measurement trace, and transfer it into your program using ASCII encoding. Binary encoding can be used for faster data transfers, as shown in the table below:

Table 6-1. Typical Trace Transfer Times (ms)

Number of Points	Binary	ASCII
51	21	47
201	23	164
401	30	314
1601	82	1200

When using binary data transfers, the entire trace is sent from the analyzer to your program in a block called a definite length block. The details of block data are described in detail in Chapter 4. The definite length block contains a header and a data section. The header indicates how many bytes are in the data section.

In order to read the definite length block, your program must first read the header, and then read the data section. Refer to the example program REALDATA in Chapter 8 for an example of how to do this.

In the REALDATA program, you will notice the following lines which read the definite block header:

```
180    ENTER @Hp8711 USING "%,A,D";A$,Digits
190    ENTER @Hp8711 USING "%,"&VAL$(Digits)&"D";Bytes
```

and these lines which read the data section:

```
200    ASSIGN @Hp8711;FORMAT OFF
210    ENTER @Hp8711;Data1(*)
```

Each measurement point in the data section is represented as 4 or 8 bytes (32 or 64 bits), depending on whether single precision or double precision numbers are requested. When using HP BASIC or IBASIC, you must select double precision numbers to match BASIC's "REAL" data type. Do this using the SCPI command `"FORM:DATA REAL,64"`. If you are using another language that supports single precision data types, you can select single precision using the SCPI command `"FORM:DATA REAL,32"`. Languages such as QuickBASIC and C have support for both single and double precision floating point numbers.

When transferring data using binary encoding, you may need to reverse the order of the bytes in each measurement point, since PCs frequently store IEEE floating point numbers with the byte order reversed. To instruct the analyzer to reverse the byte order of the data, send the command `"FORMAT:BORDer SWAPped"` before querying the trace data.

Trace Data Transfer Sizes

The following table shows how many bytes are transmitted during trace data transfers. The left column shows the format of the data, which you can specify using the SCPI command `Format:DATA`. As you can see, the size of the measurement point data and trace data varies as you change format.

Table 6-2. Size of Trace Data Transfers (in Bytes) Using the TRACE:DATA SCPI Command

Format Type (FORMat:DATA)	Type of Data	Single Measurement Point		201 Point Trace	
		Real	Complex	Real	Complex
REAL,32	IEEE 32-bit Floating Point	4	8	809	1614
REAL,64	IEEE 64-bit Floating Point	8	16	1614	3222
ASCII,5	ASCII numbers	13	26	2613	5226
ASCII,3	ASCII numbers	11	22	2211	4422
INT,16	Internal Binary	—	6	—	1212

When transmitting data in "REAL" or "INT" format, a header is sent before the data block. The header indicates the size of the data block. The header size varies in length from 3 to 11 bytes. Refer to Chapter 4 for details on the header.

Transmitting ASCII data requires no header. The ASCII values are separated by commas, and a linefeed is sent after the last value. The sizes shown in the table include the size of the comma(s) and terminating linefeed. Typical data in ASCII,5 format:

```
-1.2254E+000,+5.0035E-001,+4.5226E-001,...
```

The analyzer stores its internal data with approximately 5 significant digits of resolution. Using REAL,32 or ASCII,5 format provides sufficient precision for data transfers. However, REAL,64 may be necessary when using a programming language which does not support IEEE 32-bit floating point.

Transferring Data with IBASIC

If you are using IBASIC, your IBASIC program can avoid the overhead of using OUTPUT and ENTER to transfer trace data, and instead use the analyzer's built-in high-speed subprograms. These built-in subroutines let you quickly move data between the analyzer's measurement arrays and your program's data arrays. For example, to read the analyzer's formatted data array, use the following:

```
10 DIM Fmt(1:201)
20 INTEGER Chan
30 LOADSUB Read_fdata FROM "XFER:MEM 0,0"
40 Chan=1
50 Read_fdata(Chan,Fmt(*))
```

Refer to the *HP Instrument BASIC User's Handbook* for more details.

The table below compares the speed of IBASIC using high-speed transfer subroutines with that of a fast external controller using the SCPI `TRACE:DATA? CH1FDATA` query.

Table 6-3. Typical Trace Transfer Times (ms)

Number of Points	Controller Using Binary `TRACE:DATA?`	IBASIC Using `Read_fdata`
51	21	7
201	23	10
401	30	13
1601	82	32

Taking Sweeps

When making measurements and querying traces, your program should perform the following steps:

1. Place the analyzer's sweep in hold
2. Initiate a single sweep
3. Wait for the sweep to complete
4. Query the measurement trace

Use the following program lines perform these steps:

```
10 OUTPUT @Hp8711;"ABORT;:INIT1:CONT OFF"
20 OUTPUT @Hp8711;"INIT1"
30 OUTPUT @Hp8711;"*OPC?"
35 ENTER @Hp8711;Opc
40 OUTPUT @Hp8711;"TRACE:DATA? CH1FDATA"
45 ENTER @Hp8711;Fmt(*)
```

If you query the measurement trace while the analyzer is in continuous sweep, the query will still work, but the data may not be correct. Using INIT and *OPC? ensures that a complete sweep has finished before you query the measurement data. In many cases, you can also use the command "*WAI" in place of the "*OPC?" query, replacing lines 30 and 35 above with:

```
30 OUTPUT @Hp8711;"*WAI"
```

However, there are cases where "*WAI" will produce incorrect results. One case is when using IBASIC's high-speed subprograms to query the trace data. "*WAI" only ensures that the SCPI commands following the "*WAI" are not executed until the commands before the "*WAI" are complete. Since IBASIC subprograms don't use SCPI commands to access the trace data, "*WAI" is ineffective, and "*OPC?" should be used.

When using "*OPC?", the ENTER statement following the "*OPC?" will wait until the previous SCPI commands are complete, preventing your program from executing beyond the ENTER statement. When using "*WAI", your program can continue to run and send SCPI commands, and the analyzer will buffer them and act upon them in order.

For more details, refer to Chapter 2, "Synchronizing the Analyzer and a Controller."

CALC:DATA? versus TRACE:DATA?

The SCPI command "`CALC1:DATA?`" is functionally equivalent to the command "`TRACE:DATA? CH1FDATA`". The two can be used interchangeably for trace queries of the formatted measurement data. The "`TRACE:DATA`" command is more flexible, allowing you to query other measurement arrays and to download data to measurement arrays.

Querying Single Data Points Using Markers

If you only need to query a single data point, you can use a marker query instead of a trace query. The program segment below shows how to do this using the SCPI command CALC:MARK.

```
10   ASSIGN @Hp8711 TO 716
20   ! Take sweep here
30   OUTPUT @Hp8711;"CALC1:MARK ON"    ! turn on marker
40   OUTPUT @Hp8711;"CALC1:MARK1:X 177 MHz"   ! set frequency
50   OUTPUT @Hp8711;"CALC1:MARK1:Y?"          ! read marker
60   ENTER @Hp8711;Marker_y
70   DISP Marker_y
```

You can also use the `CALC:MARK:FUNC:RES?` query to return the results of a bandwidth search. For example:

```
10   ! Select -3 dB bandwidth
20   OUTPUT @Hp8711;"CALC:MARK:BWID -3"
30   ! Get result of bandwidth search
40   OUTPUT @Hp8711;"CALC:MARK:FUNC:RES?"
50   ENTER @Hp8711;Bwidth,Center_freq,Q,Loss
```

For more information on using markers, refer to Chapter 8, "Example Programs."

Accessing Other Measurement Arrays

The preceding sections describe how to query the formatted data array using the **TRACE:DATA?** query with the argument CH1FDATA. The formatted array is the last array in the analyzer's data processing chain, and is generally of most interest.

The analyzer also allows you to query other measurement arrays which are earlier in its data processing chain. Figure 6-2, below, shows the data processing chain.

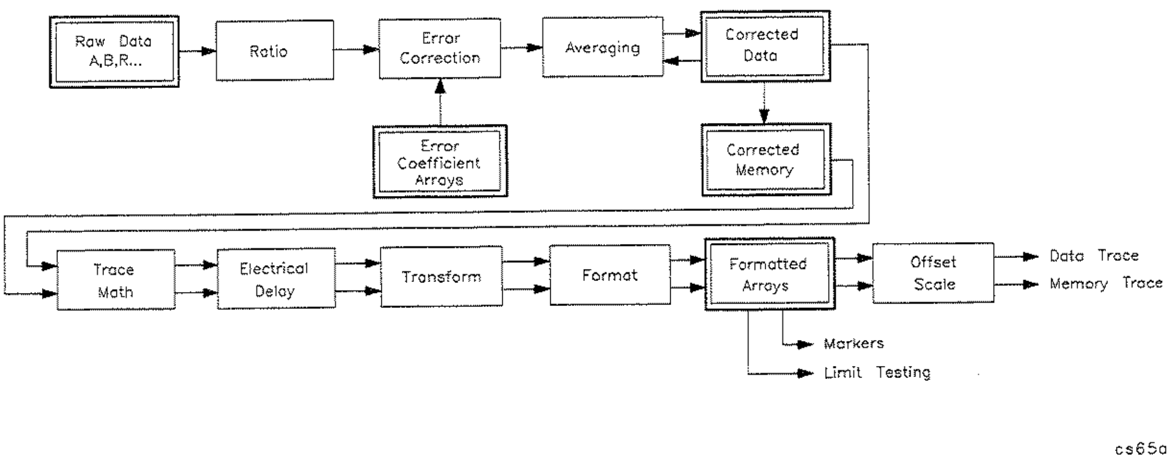


Figure 6-2. Numeric Data Flow Through the Network Analyzer

The first array is the Raw Data Array, which contains each of the separate input components (A, B, R, B*, R*, X, Y, AUX) immediately after they are measured. These arrays can be queried and set, but doing so is of limited use, since the data values contained in the arrays are uncorrected, and are not directly correlated to any meaningful reference, such as 0 dBm.

The Error Coefficient Arrays contain default correction values or values created during a measurement calibration. These arrays can be queried and set, but care should be exercised in setting them since incorrect measurements may result. If you wish to apply your own corrections in addition to the analyzer's current correction, the best technique is to use the Corrected Memory array and the Data/Memory feature, explained below.

The Corrected Data array contains the results of the currently selected measurement (Transmission, Reflection, etc.) after error correction and averaging have been applied. The measurement data in these arrays is represented as complex number pairs. When measuring the transmission response of a through cable, the magnitude of the complex numbers will be very close to 1.0. When measuring an open circuit, the magnitude of the complex numbers will be very close to 0.0. When measuring an amplifier, the magnitude of the complex numbers will be greater than 1.0.

The Corrected Memory array is filled with a copy of the Corrected Data array when the Data –> Memory operation is performed. It can be used to apply a gain correction to the measured data. This is described in the following section.

The Formatted Data array contains the measurement data after it has been formatted using the format selected by the [FORMAT] menu. Querying the Formatted Data array is described in detail at the beginning of this chapter. You can also download data to this array, and the analyzer will display the data using the current Scale and Reference values.

Applying Gain Correction Using the Memory Trace

The Corrected Memory array is filled with a copy of the Corrected Data array when the Data –> Memory operation is performed. By setting the analyzer to perform Data/Memory trace math, you can apply your own correction factor to the measurement data trace by filling the Corrected Memory array with the appropriate complex numbers.

In general, you should use the analyzer's calibration feature to correct for errors in your system. However, there may be cases where you wish to simulate the effect of adding a cable in series with your DUT, and observe how this imaginary cable will attenuate the measured response versus frequency. Or you may wish to apply an absolute offset to simulate the effect of adding or removing a pad from the measurement. These simulations are easily accomplished using the Corrected Memory array and the Data/Memory feature.

The Corrected Data and Memory arrays contain complex linear data, as opposed to logged data. When displaying the traces using Lin Mag format, the result of the Data divided by Memory operation (Data/Mem) will be to divide each point of the data trace by each point of the memory trace. When displaying data in Log Mag format, the result of Data/Memory will be equivalent to subtracting the Log Mag value of the Memory trace from that of the Data trace.

The following example BASIC code segment shows how to download a complex array from your program to the analyzer's Memory trace. The program's "Mem" array is initialized with the proper values such that when the analyzer computes Data divided by Memory, the desired increasing gain will be applied.

```
100   REAL Mem(1:201,1:2)
110   ASSIGN @Hp8711 TO 716
120   ! Fill memory array (denominator in Data/Mem)
130   ! with values that will result in an
140   ! upward sloping gain factor vs. frequency.
150   ! Used to compensate for cable loss vs. frequency
160   ! Adds 0 dB of gain at start freq; 3 dB at stop freq
170   FOR Pt=1 TO 201
180      Gain_factor_db=3.0*(Pt - 1)/200  ! 0..3 dB Power
190      Gain_factor_lin=10^(Gain_factor_db/20)
200      Mem(Pt,1)=1.0/Gain_factor_lin  ! real
210      Mem(Pt,2)=0.0                  ! imag
220   NEXT Pt
230   ! Download to the memory trace
240   OUTPUT @Hp8711;"FORM:DATA ASCII"
250   OUTPUT @Hp8711;"TRACE:DATA CH1SMEM";   ! Note the ";"
260   FOR Pt=1 TO 201
270      FOR I=1 TO 2
280         OUTPUT @Hp8711;",";Mem(Pt,I);    ! Note the ";"
290      NEXT I
300   NEXT Pt
310   OUTPUT @Hp8711;""     ! Send linefeed
320   OUTPUT @Hp8711;"CALC1:MATH (IMPL/CH1SMEM)"  ! Data/Mem
```

The example above downloads data to the corrected memory array. The data is sent by the program to the analyzer using ASCII encoding. The data is sent as ASCII characters, separated by commas. The analyzer accepts the comma separated list of numbers until it receives a linefeed to terminate the command. The program uses semicolons at the end of some OUTPUT statements to avoid sending a linefeed until all of the data has been sent. After the last number is sent, the program sends a linefeed, and the analyzer accepts the data.

Remember, for faster transfers, use binary data encoding instead of ASCII.

Performing Your Own Data Processing

After the analyzer has made a measurement, you can read the measurement trace and perform your own post-processing on it, and display the result on the screen. This is done using these steps:

1. Initiate a sweep
2. Wait for the sweep to finish
3. Read the measurement data into an array in your program
4. Perform your post-processing on the measurement data
5. Write (download) the post-processed data to the analyzer's memory trace.

You may want to instruct the analyzer to display only the memory trace and not the data trace, so that only your post-processed data is seen.

The program below demonstrates how to perform data post-processing. It takes the measurement data and reverses it, such that the low frequency data is displayed on the right end of the trace, and the high frequency data is displayed on the left.

```
100   ! Display the measurement data backwards
110   REAL Fmt(1:201)
120   ASSIGN @Hp8711 TO 716
130   !
140   OUTPUT @Hp8711;"FORM:DATA ASCII"
150   OUTPUT @Hp8711;"ABOR;INIT:CONT OFF;*WAI"
160   OUTPUT @Hp8711;"DISP:WIND:TRAC1 OFF;TRAC2 ON"
170   LOOP
180      ! Take sweep
190        OUTPUT @Hp8711;"INIT1;*WAI"
200        ! Read the trace from the formatted data array
210        OUTPUT @Hp8711;"TRACE:DATA? CH1FDATA"
220        ENTER @Hp8711;Fmt(*)
230        ! Download the trace, backwards,
235        ! to the formatted memory array
240        OUTPUT @Hp8711;"TRACE:DATA CH1FMEM";    ! Note the ";"
250        FOR Pt=1 TO 201
260            OUTPUT @Hp8711;",";Fmt(202-Pt);     ! Note the ";"
270        NEXT Pt
280        OUTPUT @Hp8711;""    ! Send linefeed
290   END LOOP
```

This example program uses ASCII trace data transfers. Higher speed can be achieved using binary data transfers. If using IBASIC, high-speed subroutines can be used for even greater performance. Refer to the IBASIC Handbook for details.

Downloading Trace Data Using Binary Encoding

Data traces can be downloaded to the analyzer using binary encoding. Using binary encoding is faster than using ASCII encoding. As mentioned in Chapter 4, the binary encoded trace is transferred as a block; the block containing a header and a data section. There are two different types of blocks that can be used: a definite length block, and an indefinite length block.

To send trace data using a definite length block, your program must calculate the number of bytes in the data segment of the block, and create a block header which tells the analyzer how many bytes are in the data segment.

For example, if you are sending a trace with 201 data points and using 64-bit floating point numbers for each data point ("FORM:DATA REAL,64"), the block's data segment will contain 1608 bytes (201 points * 8 bytes/point). The header characters for a 1608 byte block are: "#41608". The first digit after the "#", "4" tells how many following digits are used to specify the size. In this case, 4 digits follow, and those digits are "1608", meaning that the block contains 1608 bytes.

For example:

```
TRAC CH1FDATA,#41608<binary_data_starts_here>
```

When you send a definite length block to the analyzer, the analyzer will will read the data segment bytes, stopping when it receives the number specified in the block header.

To send trace data using an indefinite length block, your program simply sends a block header of "#0", followed by the data segment. After sending the data segment, your program must terminate the data block by sending an EOI. The analyzer will read the data segment bytes, stopping when it receives an EOI. To send an EOI using BASIC, you can use the statement:

```
OUTPUT @Hp8711;END
```

Internal Measurement Arrays

The following sections describe the sequence of math operations and the resulting data arrays as the measurement information flows from the raw data arrays to the display. This information explains the measurement arrays accessible via HP-IB.

Figure 6-3 is a data processing flow diagram that represents the flow of numerical data. The data passes through several math operations, denoted in the figure by single-line boxes. Most of these operations can be selected and controlled with the front panel CONFIGURE block menus. The data is stored in arrays along the way, denoted by double-line boxes. These arrays are places in the flow path where data is accessible via HP-IB. While only a single flow path is shown, two identical paths are available, corresponding to measurement channels 1 and 2.

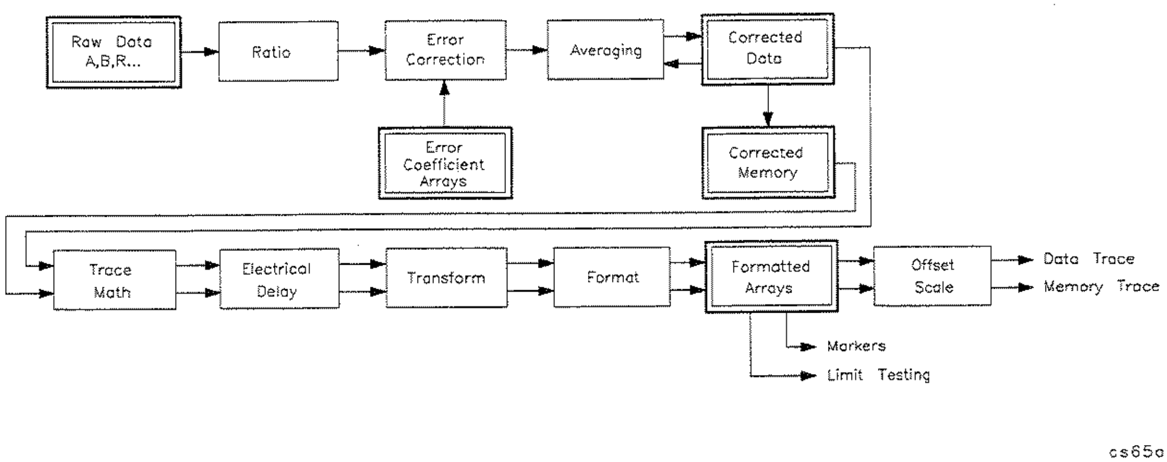


Figure 6-3. Numeric Data Flow Through the Network Analyzer

Raw Data Arrays

These arrays are linear measurements of the inputs used in the selected measurement. Note that these numbers are complex pairs. These arrays are directly accessible via HP-IB and referenced as CH[1|2]AFWD, CH[1|2]BFWD and CH[1|2]RFWD.

Table 6-4. Raw Data Arrays

Selected Measurement	Raw Arrays
Transmission (B/R)	B = CH[1|2]BFWD, R = CH[1|2]RFWD
Reflection (A/R)	A = CH[1|2]AFWD, R = CH[1|2]RFWD
A	A = CH[1|2]AFWD
B	B = CH[1|2]BFWD
R	R = CH[1|2]RFWD
Power (B*)	B* = CH[1|2]BFWD
Conversion Loss (B*/R*)	B* = CH[1|2]BFWD, R* = CH[1|2]RFWD
R*	R* = CH[1|2]RFWD
AM Delay (Y/X)	Y = CH[1|2]BFWD, X = CH[1|2]RFWD
X	X = CH[1|2]RFWD
Y	Y = CH[1|2]BFWD
Y/R*	Y = CH[1|2]BFWD, R* = CH[1|2]RFWD
Y/X, X/Y	Y = CH[1|2]BFWD, X = CH[1|2]RFWD

NOTE

Raw data for AUX INPUT is not available via HP-IB. Use the corrected data array to access AUX INPUT data.

Ratio Calculations

These are performed if the selected measurement is a ratio (e.g. A/R or B/R). This is simply a complex divide operation. If the selected measurement is absolute (e.g. A or B), no operation is performed.

Error Correction

Error correction is performed next if correction is turned on. Error correction removes repeatable systematic errors (stored in the error coefficient arrays) from the raw arrays. The operations performed depend on the selected measurement type.

Error Coefficient Arrays

The error coefficient arrays are either default values or are created during a measurement calibration. These are used whenever correction is on. They contain complex number pairs, and are accessible via HP-IB and are referenced as `CH[1|2]SCORR1`, `CH[1|2]SCORR2`, `CH[1|2]SCORR3` and `CH[1|2]SCORR4`.

Table 6-5. Error Coefficient Arrays

Selected Measurement	Error Coefficient Arrays
Transmission (B/R) Response	CH[1|2]SCORR1 = Tracking
Transmission (B/R) Response & Isolation	CH[1|2]SCORR1 = Tracking CH[1|2]SCORR2 = Isolation Term
Transmission (B/R) Enhanced Response	CH[1|2]SCORR1 = Directivity CH[1|2]SCORR2 = Source Match CH[1|2]SCORR3 = Reflection Tracking CH[1|2]SCORR4 = Transmission Tracking
Reflection (A/R)	CH[1|2]SCORR1 = Directivity CH[1|2]SCORR2 = Source Match CH[1|2]SCORR3 = Tracking
Broadband Internal	CH[1|2]SCORR1 = R* Response

NOTE

These arrays do not apply to Broadband External measurements.

Averaging

Averaging is a noise reduction technique. This calculation involves taking the complex exponential average of several consecutive sweeps. This averaging calculation is different than the System Bandwidth setting. System Bandwidth uses digital filtering, applying noise reduction to the measured data before it is stored into the Raw Data Arrays.

Corrected Data Arrays

The combined results of the ratio, error correction and averaging operations are stored in the corrected data arrays as complex number pairs. These arrays are accessible via HP-IB and referenced as `CH[1|2]SDATA`.

Corrected Memory Arrays

If the Data–>Mem or Normalize operations are performed, the corrected data arrays are copied into the corrected memory arrays. These arrays are accessible via HP-IB and referenced as `CH[1|2]SMEM`.

Trace Math Operation

This selects either the corrected data array, or the corrected memory array, or both to continue flowing through the data processing path. In addition, the complex ratio of the two (Data/Memory) can also be selected. If memory is displayed, the data from the memory arrays goes through exactly the same data processing flow path as the data from the data arrays.

⊛ Electrical Delay

This block adds or subtracts phase, based on the settings of Phase Offset, Electrical Delay, and Port Extension. The Electrical Delay and Port Extension features add or subtract phase in proportion to frequency. This is equivalent to "line stretching" or artificially moving the measurement reference plane. (See the *HP 8712C/14C User's Guide* for more details on these features.)

Transform (Option 100 only)

This block converts frequency domain data into distance domain, or into an SRL impedance value when measuring fault location or SRL. The transform employs an inverse fast Fourier transform (FFT) algorithm to accomplish the conversion.

Formatting

This converts the complex number pairs into a scalar representation for display, according to the selected format (e.g. Log Mag, SWR, etc). These formats are often easier to interpret than the complex number representation. Note that after formatting, it is impossible to recover the complex data.

Formatted Arrays

The results so far are stored in the formatted data and formatted memory arrays. It is important to note that marker values and marker functions are all derived from the formatted arrays. Limit testing is also performed on the formatted arrays. These arrays are accessible via HP-IB and referenced as `CH[1|2]FDATA` and `CH[1|2]FMEM`.

Offset and Scale

These operations prepare the formatted arrays for display. This is where the reference position, reference value, and scale calculations are performed, as appropriate for the format.

7

Using Graphics

Using Graphics

The analyzer has a set of user graphics commands that can be used to create graphics and messages on the display. The GRAPHICS example program uses some of these commands to draw a simple setup diagram. These commands, listed below, are of the form:

```
DISPlay:WINDow[1|2|10]:GRAPhics:<mnemonic>.
```

The number specified in the WINDow part of the command selects where the graphics are to be written.

WINDow1 draws the graphics to the channel 1 measurement screen. (This is the default if no window is specified in the mnemonic.)

WINDow2 draws the graphics to the channel 2 measurement screen.

WINDow10 draws the graphics to an IBASIC display partition. (This window is only available on instruments with IBASIC — Option 1C2.)

NOTE

When graphics commands are used to write directly to a measurement screen they write to the static graphics plane (the same plane where the graticule is drawn). There is no sweep-to-sweep speed penalty once the graphics have been drawn.

Unless otherwise specified, the graphics commands listed below start at the current pen location. All sizes are dimensioned in pixels.

`DISPlay:WINDow[1|2|10]:GRAPhics:CIRCle <y_radius>`

`DISPlay:WINDow[1|2|10]:GRAPhics:CLEar`

`DISPlay:WINDow[1|2|10]:GRAPhics:COLor <pen>`

- color choices are: 0 for erase, 1 for bright, 2 for dim

`DISPlay:WINDow[1|2|10]:GRAPhics[:DRAW] <new_x>,<new_y>`

`DISPlay:WINDow[1|2|10]:GRAPhics:LABel <string>`

`DISPlay:WINDow[1|2|10]:GRAPhics:LABel:FONT <font>`

- font choices are: SMALl, HSMall, NORMal, HNORmal, BOLD, HBOLd, SLANt, HSLant
  (H as the first letter of the font name indicates highlighted text - inverse video).

`DISPlay:WINDow[1|2|10]:GRAPhics:MOVE <new_x>,<new_y>`

`DISPlay:WINDow[1|2|10]:GRAPhics:RECTangle <width>,<height>`

`DISPlay:WINDow[1|2|10]:GRAPhics:SCALe <xmin>,<xmax>,<ymin>,<ymax>`

`DISPlay:WINDow[1|2|10]:GRAPhics:STATe?`

Window Geometry

Even though there are only three graphics windows, these windows can have different sizes and locations.

The size and location of the graphics window are determined by the display configuration currently in use — split screen measurements, full screen measurements, and full or partial IBASIC display partitions will affect the dimensions of the graphics window in use.

The sizes of the different graphics windows are listed below. Figure 7-1 shows the IBASIC display partitions.

- Measurement channel 1 or 2 full screen measurement: width=501 pixels, height=401 pixels.
- Measurement channel 1 or 2 split screen measurement: width=501 pixels, height=201 pixels.
- IBASIC full screen display: width=537 pixels, height=439 pixels.
- IBASIC upper display: width=537 pixels, height=199 pixels.
- IBASIC lower display: width=537 pixels, height=197 pixels.

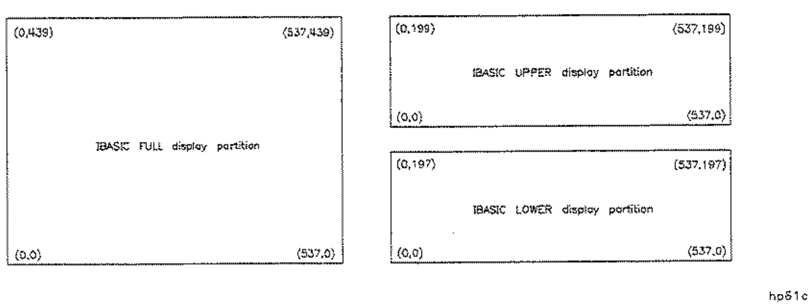


Figure 7-1. Pixel Dimensions with Available Display Partitions

There is a set of queries that can be used to determine the size and location of the display window in use.

These queries, listed below, return the width and height of the window or the absolute location of its lower left or upper right corners. All the coordinates and sizes are dimensioned in pixels.

```
DISPlay:WINDow[1|2|10]:GEOMetry:LLEFt?
DISPlay:WINDow[1|2|10]:GEOMetry:SIZE?
DISPlay:WINDow[1|2|10]:GEOMetry:URIGht?
```

NOTE

The origin of EVERY graphics window is its lower left corner. The locations returned in response to the `LLEFt` and `URIGht` are relative to the ABSOLUTE origin of the entire display, NOT to the graphics window.

The Graphics Buffer

The analyzer has a graphics buffer that is used to refresh the graphics display if needed. When the buffer is full, additional graphics can still be drawn — BUT they will not be refreshed. The graphics buffer can be turned on and off using the following command (which is used in the `GRAPHICS` example program).

```
DISPlay:WINDow:GRAPhics:BUFFer[:STATe] <ON|OFF>
```

The graphics buffer will hold up to:

500 lines
40 circles
40 rectangles
50 strings (60 characters long)

Use the following command to clear the graphics buffer and user-graphics display.

```
DISPlay:WINDow:GRAPhics:CLEar
```

NOTE

Only graphics that can be refreshed will be printed or plotted. If you intend to print or plot your graphics, make sure they will fit within the graphics buffer.

8

Example Programs

Example Programs

Most of the example programs listed in this manual are written in HP BASIC. They are also compatible with IBASIC (HP Instrument BASIC). An optional internal controller can be purchased with your analyzer (Option 1C2). This controller runs IBASIC directly on the analyzer. It controls the analyzer over an internal interface bus that operates the same way as the external HP-IB interface. For more information about IBASIC refer to the *HP Instrument BASIC User's Handbook*.

The example programs are provided on two disks that are included with the network analyzer. Both disks contain the same examples which are written mainly in HP BASIC; only the disk format is different. The analyzer's internal 3.5" disk drive is designed to be DOS compatible, however, it can read from LIF formatted disks. Therefore, either disk can be used to supply programs for the analyzer's internal IBASIC controller.

Example Programs Disk – DOS Format HP part number 08712-10019

Example Programs Disk – LIF Format HP part number 08712-10021

Because the examples are designed to run in different environments, the setup at the beginning of each program must determine the operating environment and properly set the analyzer's HP-IB address. In these examples, the internal IBASIC controller uses the address 800 when communicating with the analyzer (the internal HP-IB is at select code 8). The default address of 716 is used when the programs are being run on an external controller.

A version of the following lines is included in most of the example programs. The use of the `Internal` (internal-controller) flag varies due to differences in the programs needs.

`10 IF POS(SYSTEM$("SYSTEM ID"),"HP 87") THEN`	*Identify the operating system.*
`20 ASSIGN @Hp8711 TO 800`	*If internal, set address to 800.*
`30 Internal=1`	*Set internal-control flag to 1.*
`40 ELSE`	*If external, set address to 716.*
`50 ASSIGN @Hp8711 TO 716`	
`60 Internal=0`	*Set internal-control flag to 0.*
`70 ABORT 7`	*Abort all bus transactions and give active control of the bus to the computer.*
`80 CLEAR 716`	*Send a selected device clear (SDC) to the analyzer – this clears all HP-IB errors, resets the HP-IB interface and clears syntax errors. (It does not affect the status reporting system.)*
`90 END IF`	

NOTE

The example programs on the disks that were shipped with your analyzer may not appear exactly as listed in this chapter. The programs on the disks are the most up-to-date versions of each program. Also, check your disk listings for new programs that may not be listed here.

The following table shows the sections and example programs that are contained in this chapter:

Section Title	Example Program	Program Description
Configuring Measurements	SETUP	Sets up a basic measurement, demonstrates use of *WAI
	LIMITEST	Performs automatic pass/fail testing with limit lines
	POWERSWP	Performs a power sweep measurement
Transfer of Data to/from the Analyzer	MARKERS	Transfers data using markers
	SMITHMKR[1]	Measures reflection of a filter in Smith chart and polar formats
	ASCDATA	Transfers data using ASCII format
	REALDATA	Transfers data using the IEEE 64-bit floating point REAL format
	INTDATA	Transfers data using the 16-bit INTEGER format
	FAST_CW	Transfers marker data in CW sweep mode
Calibration	TRANCAL	Performs a transmission calibration
	REFLCAL	Performs a reflection calibration
	LOADCALS	Uploads and downloads correction arrays
	CALKIT	This is *not* a program, it is an instrument state file for downloading user-defined cal kit definitions
Instrument State and Save/Recall	LEARNSTR	Uses the learn string to upload and download instrument states
	SAVERCL	Saves and recalls instrument states, calibrations and data
Hardcopy Control	PRINTPLT	Uses the serial and parallel ports for hardcopy output
	PASSCTRL	Uses pass control and the HP-IB for hardcopy output
	FAST_PRT	Provides fast graph dumps to PCL5 printers
Service Request	SRQ	Generates a service request interrupt
	SRQ_INT	Monitors the status report of the analyzer
File Transfer Over HP-IB	GETFILE	Transfers a file from the analyzer to an external controller
	PUTFILE	Transfers a file from an external controller to the analyzer

1 For use with HP 8712C and 8714C only

Section Title	Example Program	Program Description
Customized Display	GRAPHICS	Uses graphics and softkeys to create customized procedures
	GRAPH2	Draws an instrument and DUT onto the display
	GETPLOT	Demonstrates how to read an HPGL graphics file
Annotation	USERANOT	Demonstrates how to use user-defined annotation
	FREQBLNK	Demonstrates how to conceal sensitive frequency information
	KEYCODES	Reads key presses and knob positions from the analyzer
Marker Functions	MKR_MATH	Demonstrates how to program marker math functions (statistics, flatness, and RF filter stats)
Marker Limit Testing	LIM_FLAT	Demonstrates how to test for a marker flatness limit
	LIM_PEAK	Demonstrates how to test for a marker peak-to-peak ripple limit
	LIM_MEAN	Demonstrates how to test for a marker mean limit
SRL Measurements[1]	MEAS_SRL	Demonstrates the effects of various connector modeling on a structural return loss (SRL) measurement
	SRL_SRQ	Initiates an SRL cable scan, then uses the analyzer's status model to trigger an SRQ when the cable scan has completed
Fault Location Measurements[1]	FAULT	Demonstrates the effects of various fault location frequency modes on a cable measurement
	USR_FLOC	Shows how fault location measurements can be simplified by using the **User BEGIN** key[2]

1 Option 100 only

2 You must have Option 1C2, IBASIC, installed to use the "User BEGIN" function

Section Title	Example Program	Program Description
Multiport Test Set Measurements[1]	PORT_SEL	Uses graphics to show internal connections of the HP 87075C when different ports are selected
	TSET_CAL	Attempts to recall "TSET_CAL.CAL" from non-volatile RAM or the internal disk drive. If successful, invokes the recalled test set cal for transmission and reflection of measurement channels 1 and 2
TTL Output	TTL_IO	Monitors the user TTL bit on rear panel
AM Delay[2]	AMDELAY	Demonstrates the calibration and AM delay measurement of a bandpass filter
LAN Usage[3]	See the *Option 1F7 User's Guide* for listings and information on example programs for LAN usage.	

1 For use with the HP 87075C multiport test set.

2 Options 1DA or 1DB only.

3 Option 1F7 only.

Configuring Measurements

SETUP	Setting up a basic measurement. The example also demonstrates the use of the `*WAI` command.
LIMITEST	Performing automatic PASS/FAIL testing with limit lines. The example also demonstrates some methods of combining mnemonics for more efficient programming.
POWERSWP	Setting up a power sweep measurement.

SETUP Example Program

This program demonstrates how to set up the analyzer to make a basic measurement. The *WAI command is used extensively throughout this program. This has the effect of making sure that the commands are executed in the order they are received. More information about making measurements with the analyzer is available in your analyzer's *User's Guide*.

```
1000 !Filename:  SETUP
1010 !
1020 ! Description:
1030 !   Set Channel 1 to measure filter's transmission.
1040 !   Set Channel 2 to measure filter's reflection
1050 !   Prompt user for start and stop freq, and set them.
1060 !   Take a sweep.
1070 !   Set Scale and Reference levels.
1080 !
1090 !
1100 COM /Sys_state/ @Hp87xx,Scode
1110 ! Identify I/O Port
1120 CALL Iden_port
1130 !
1140 !
1150 ! Preset the instrument.
1160 OUTPUT @Hp87xx;"SYST:PRES;*WAI"
1170 !
1180 ! Configure the analyzer to measure transmission
1190 ! of a filter on channel 1.  This is the command
1200 ! for the BEGIN Filter Transmissn key sequence.
1210 OUTPUT @Hp87xx;"CONF 'FILT:TRAN';*WAI"
1220 !
1230 ! Put the instrument in trigger hold mode.
1240 OUTPUT @Hp87xx;"ABOR;:INIT:CONT OFF;*WAI"
1250 !
1260 ! Turn on channel 2.
1270 OUTPUT @Hp87xx;"SENS2:STAT ON;*WAI"
1280 !
1290 ! Configure channel 2 to measure reflection.  This
1300 ! is the command for the CHAN 2 Reflection key sequence.
```

```
1310 OUTPUT @Hp87xx;"SENS2:FUNC 'XFR:POW:RAT 1,0';DET NBAN"
1320 !
1330 ! Wait for the previous commands to complete execution
1340 ! (respond to the *OPC?).
1350 OUTPUT @Hp87xx;"*OPC?"
1360 ENTER @Hp87xx;Opc
1370 !
1380 ! Input a start frequency.
1390 INPUT "Enter Start Frequency (MHz):",Start_f
1400 !
1410 ! Input a stop frequency.
1420 INPUT "Enter Stop Frequency (MHz):",Stop_f
1430 !
1440 ! Set the start and stop frequencies of the analyzer
1450 ! to the values entered.
1460 OUTPUT @Hp87xx;"SENS2:FREQ:STAR";Start_f;"MHz;STOP";Stop_f;"MHz;*WAI"
1470 !
1480 ! Trigger a single sweep.
1490 OUTPUT @Hp87xx;"INIT;*OPC?"
1500 !
1510 ! Wait for the sweep to be completed.
1520 ENTER @Hp87xx;Opc
1530 !
1540 ! Set up the scale and reference parameters for channel 1.
1550 OUTPUT @Hp87xx;"DISP:WIND1:TRAC:Y:PDIV 10 DB;RLEV 0 DB;RPOS 8"
1560 !
1570 ! Now for channel 2.
1580 OUTPUT @Hp87xx;"DISP:WIND2:TRAC:Y:PDIV 5 DB;RLEV 0 DB;RPOS 8"
1590 !
1600 ! Make channel 1  active (transmission)
1610 OUTPUT @Hp87xx;"SENS1:STAT ON"
1620 !
1630 ! Display the current start and stop frequencies.
1640 DISP "Done measuring.  Start =";Start_f;"MHz     Stop =";Stop_f;"MHz"
1650 END
1660 !
```

```
1670 !************************************************************
1680 ! Iden_port:   Identify io port to use.
1690 ! Description: This routines sets up the I/O port address for
1700 !              the SCPI interface.  For "HP 87xx" instruments,
1710 !              the address assigned to @Hp87xx = 800 otherwise,
1720 !              716.
1730 !************************************************************
1740 SUB Iden_port
1750     COM /Sys_state/ @Hp87xx,Scode
1760 !
1770     IF POS(SYSTEM$("SYSTEM ID"),"HP 87")<>0 THEN
1780         ASSIGN @Hp87xx TO 800
1790         Scode=8
1800     ELSE
1810         ASSIGN @Hp87xx TO 716
1820         Scode=7
1830     END IF
1840 !
1850 SUBEND !Iden_port
1860 !
```

LIMITEST Example Program

This program demonstrates how to set up and use limit lines over the HP-IB. The example device used in this program is the demonstration filter that is shipped with the analyzer. The program sets up the basic measurement, downloads the limit lines and uses the status registers to determine of the device passes its specifications. For more information about limit lines, refer to the *User's Guide.* For information about using the status registers, refer to the previous section "Using the Status Registers."

This example also demonstrates how multiple command mnemonics can be combined together. The easiest commands to combine are ones that are closely related on the command tree (such as the start and stop frequency of a limit segment). For more information of command mnemonics, refer to Chapter 10, "Introduction to SCPI."

```
1000 !Filename:  LIMITEST
1010 !
1020 DIM Title$[30]
1030 !
1040 !
1050 COM /Sys_state/ @Hp87xx,Scode
1060 ! Identify I/O Port
1070 CALL Iden_port
1080 !
1090 ! Perform a system preset; this clears the limit table.
1100 OUTPUT @Hp87xx;"SYST:PRES;*WAI"
1110 !
1120 ! Set up the source frequencies for the measurement.
1130 OUTPUT @Hp87xx;"SENS1:FREQ:STAR 10 MHZ;STOP 400 MHZ;*WAI"
1140 !
1150 ! Set up the receiver for the measurement parameters
1160 ! (Transmission in this case).
1170 OUTPUT @Hp87xx;"SENS1:FUNC 'XFR:POW:RAT 2,0';DET NBAN;*WAI"
1180 !
1190 ! Configure the display so measurement
1200 ! results are easy to see.
1210 OUTPUT @Hp87xx;"DISP:WIND1:TRAC:Y:PDIV 10 DB;RLEV 0 DB;RPOS 9"
1220 !
```

```
1230 ! Reduce the distractions on the display by
1240 ! getting rid of notation that will not be
1250 ! needed in this example.
1260 OUTPUT @Hp87xx;"DISP:ANN:YAX OFF"
1270 !
1280 ! Erase the graticule grid for the same reason.
1290 OUTPUT @Hp87xx;"DISP:WIND1:TRAC:GRAT:GRID OFF"
1300 !
1310 ! Create and turn on the first segment for
1320 ! the new limit lines; this one is a maximum
1330 ! limit.
1340 OUTPUT @Hp87xx;"CALC1:LIM:SEGM1:TYPE LMAX;STAT ON"
1350 !
1360 ! Set the amplitude limits for the first limit
1370 ! segment.
1380 OUTPUT @Hp87xx;"CALC1:LIM:SEGM1:AMPL:STAR -70;STOP -70"
1390 !
1400 ! Set the frequency of the first limit segment.
1410 OUTPUT @Hp87xx;"CALC1:LIM:SEGM1:FREQ:STAR 10 MHZ;STOP 75 MHZ"
1420 !
1430 ! Create and turn on a second maximum limit
1440 ! segment.
1450 OUTPUT @Hp87xx;"CALC1:LIM:SEGM2:TYPE LMAX;STAT ON"
1460 !
1470 ! Set the amplitude limits for segment 2.
1480 OUTPUT @Hp87xx;"CALC1:LIM:SEGM2:AMPL:STAR 0;STOP 0"
1490 !
1500 ! Set the frequency range for segment 2.
1510 OUTPUT @Hp87xx;"CALC1:LIM:SEGM2:FREQ:STAR 145 MHZ;STOP 200 MHZ"
1520 !
1530 ! Create and turn on a third limit segment;
1540 ! this one is a minimum limit.
1550 OUTPUT @Hp87xx;"CALC1:LIM:SEGM3:TYPE LMIN;STAT ON"
1560 !
1570 ! Set the amplitude limits for segment 3.
1580 OUTPUT @Hp87xx;"CALC1:LIM:SEGM3:AMPL:STAR -6;STOP -6"
1590 !
1600 ! Set the frequency range for segment 3.
1610 OUTPUT @Hp87xx;"CALC1:LIM:SEGM3:FREQ:STAR 150 MHZ;STOP 195 MHZ"
1620 !
1630 ! Create and set parameters for segment 4.
```

```
1640 OUTPUT @Hp87xx;"CALC1:LIM:SEGM4:TYPE LMAX;STAT ON"
1650 OUTPUT @Hp87xx;"CALC1:LIM:SEGM4:AMPL:STAR -60;STOP -60"
1660 OUTPUT @Hp87xx;"CALC1:LIM:SEGM4:FREQ:STAR 290 MHZ;STOP 400 MHZ"
1670 !
1680 ! Send an operation complete query to ensure that
1690 ! all overlapped commands have been executed.
1700 OUTPUT @Hp87xx;"*OPC?"
1710 !
1720 ! Wait for the reply.
1730 ENTER @Hp87xx;Opc
1740 !
1750 ! Turn on the display of the limit lines.
1760 OUTPUT @Hp87xx;"CALC1:LIM:DISP ON"
1770 !
1780 ! Turn on the pass/fail testing; watch the
1790 ! analyzer's display for the pass/fail indicator.
1800 OUTPUT @Hp87xx;"CALC1:LIM:STAT ON"
1810 !
1820 ! Take a controlled sweep to ensure that
1830 ! there is real data present for the limit test.
1840 OUTPUT @Hp87xx;"ABOR;:INIT1:CONT OFF;:INIT1;*WAI"
1850 !
1860 ! Query the limit fail condition register to see
1870 ! if there is a failure.
1880 OUTPUT @Hp87xx;"STAT:QUES:LIM:COND?"
1890 !
1900 ! Read the register's contents.
1910 ENTER @Hp87xx;Fail_flag
1920 !
1930 ! Bit 0 is the test result for channel 1 while
1940 ! bit 1 is the results for channel 2 limit testing.
1950 IF BIT(Fail_flag,0)=1 THEN
1960 !
1970 ! In case of failure, give additional direction
1980 ! to the operator using the title strings.
1990     Title$="Limit Test FAIL - Tune device"
2000 !
2010 ! Turn on the title string.
2020     OUTPUT @Hp87xx;"DISP:ANN:TITL1:DATA '"&Title$&"';STAT ON"
2030 !
2040 ! Turn on continuous sweep mode for tuning.
```

```
2050     OUTPUT @Hp87xx;"INIT1:CONT ON;*WAI"
2060 !
2070 ! Loop while the tuning is taking place.
2080     LOOP
2090 !
2100 ! Monitor the status of the limit fail
2110 ! condition register.
2120         OUTPUT @Hp87xx;"STAT:QUES:LIM:COND?"
2130         ENTER @Hp87xx;Fail_flag
2140 !
2150 ! Check the limit fail bit.  Exit if the
2160 ! device has been tuned to pass the test.
2170     EXIT IF BIT(Fail_flag,0)=0
2180     END LOOP
2190 END IF
2200 !
2210 ! Turn off the prompt to the operator and
2220 ! return the analyzer to the continuously
2230 ! sweeping mode.
2240 OUTPUT @Hp87xx;"DISP:ANN:TITL1 OFF;:INIT:CONT ON;*WAI"
2250 END
2260 !
2270 !**************************************************************
2280 ! Iden_port:   Identify io port to use.
2290 ! Description: This routines sets up the I/O port address for
2300 !              the SCPI interface.  For "HP 87xx" instruments,
2310 !              the address assigned to @Hp87xx = 800 otherwise,
2320 !              716.
2330 !**************************************************************
```

```
2340 SUB Iden_port
2350     COM /Sys_state/ @Hp87xx,Scode
2360 !
2370     IF POS(SYSTEM$("SYSTEM ID"),"HP 87")<>0 THEN
2380         ASSIGN @Hp87xx TO 800
2390         Scode=8
2400     ELSE
2410         ASSIGN @Hp87xx TO 716
2420         Scode=7
2430     END IF
2440 !
2450 SUBEND !Iden_port
2460 !
```

POWERSWP Example Program

This program demonstrates how to set up a power sweep. It shows how to query the instrument to determine its power sweep ranges, how to place a marker at a given stimulus power value (x-axis), how to read the measured power at a marker (y-axis) and how to read an entire trace of power sweep data.

```
1000 ! Filename:  POWERSWEEP
1010 ! _
1020 !
1030 !  Description: Query the power sweep ranges,
1040 !  take a power sweep, and use markers to read
1050 !  gain at marker.  Then query the trace.
1060 ! _
1070 DIM Trace_pwr(1:201)
1080 !
1090 COM /Sys_state/ @Hp87xx,Scode
1100 ! Identify I/O Port
1110 CALL Iden_port
1120 !
1130 !
1140 !---------------------------------------------------
1150 ! Initialize the 871x; set to power sweep mode.
1160 ! Set CW mode and freq.
1170 !
1175 OUTPUT @Hp87xx;"SYST:PRESET;*OPC?"
1176 ENTER @Hp87xx;Opc
1180 OUTPUT @Hp87xx;"SENS1:FUNC 'XFR:POW 2';DET BBAN;*WAI"
1190 OUTPUT @Hp87xx;"SENS1:FREQ:CENT 1 GHZ;*WAI"
1200 ! Note that CW mode is set before Power-sweep mode
1210 OUTPUT @Hp87xx;"DISP:ANN:FREQ1:MODE CW;:SENS:FREQ:SPAN 0 HZ;*WAI"
1220 OUTPUT @Hp87xx;"POWER:MODE SWEEP;*WAI"
1230 !
1240 !---------------------------------------------------
1250 ! Determine the Min/Max power settings for each
1260 ! attenuator range.
1270 !
1280 FOR Atten=0 TO 60 STEP 10
```

```
1290      OUTPUT @Hp87xx;"SOUR:POW:RANG ATT"&VAL$(Atten)&";*WAI"
1300      OUTPUT @Hp87xx;"SOUR:POW:STAR? MIN"
1310      ENTER @Hp87xx;Pwr_min
1320      OUTPUT @Hp87xx;"SOUR:POW:STAR? MAX"
1330      ENTER @Hp87xx;Pwr_max
1340      PRINT "Atten: ";Atten;" Min: ";Pwr_min;" Max: ";Pwr_max
1350 NEXT Atten
1360 !
1370 !---------------------------------------------------
1380 ! Find the optimum power sweep range,
1390 ! defined as being that range for which either:
1400 ! 1) Both the desired Start and Stop Power levels may be set,
1410 ! 2) The desired Start Power may be set and the
1420 !    Power Range is maximized.
1425 ! Then, modify next 3 lines of code to get desired settings.
1430 !---------------------------------------------------
1440 ! Set Start and Stop power levels for power sweep.
1450 !
1460 OUTPUT @Hp87xx;"SOUR:POW:RANG ATT0;*WAI"
1470 OUTPUT @Hp87xx;"SOUR:POW:STAR -2 DBM;*WAI"
1480 OUTPUT @Hp87xx;"SOUR:POW:STOP 4 DBM;*WAI"
1490 ! Take one sweep, wait till done
1500 OUTPUT @Hp87xx;"ABOR;:INIT1:CONT OFF;:INIT1;*OPC?"
1510 ENTER @Hp87xx;Opc
1520 !---------------------------------------------------
1530 ! Read marker, display power in, power out, gain.
1540 ! Note that the
1550 !   X-axis is swept output power from the source,
1560 !   Y-axis is power measured by the receiver.
1570 !
1580 OUTPUT @Hp87xx;"CALC1:MARK1 ON"
1590 ! Set marker to start power, wait till done.
1600 OUTPUT @Hp87xx;"CALC1:MARK:X -2;*OPC?"
1610 ENTER @Hp87xx;Opc
1620 ! Read Marker Source power level and measured power.
1630 OUTPUT @Hp87xx;"CALC1:MARK1:X?"
1640 ENTER @Hp87xx;Pwr_src
1650 OUTPUT @Hp87xx;"CALC1:MARK1:Y?"
1660 ENTER @Hp87xx;Pwr_meas
1670 ! Read entire trace array.
1680 OUTPUT @Hp87xx;"FORM:DATA ASC,3"
```

```
1690 OUTPUT @Hp87xx;"TRAC? CH1FDATA"
1700 ENTER @Hp87xx;Trace_pwr(*)
1710 !
1720 PRINT "Source Power @ Marker = "&VAL$(Pwr_src)&"dBm"
1730 PRINT "Received Power @ Marker = "&VAL$(Pwr_meas)&"dBm"
1740 PRINT "Gain @ Marker = "&VAL$(Pwr_meas-Pwr_src)&"dB"
1750 PRINT "Power Sweep Trace Point #1:  "&VAL$(Trace_pwr(1))&"dBm"
1760 END
1770 !
1780 !****************************************************************
1790 ! Iden_port:   Identify io port to use.
1800 ! Description: This routines sets up the I/O port address for
1810 !              the SCPI interface.  For "HP 87xx" instruments,
1820 !              the address assigned to @Hp87xx = 800 otherwise,
1830 !              716.
1840 !****************************************************************
1850 SUB Iden_port
1860     COM /Sys_state/ @Hp87xx,Scode
1870 !
1880     IF POS(SYSTEM$("SYSTEM ID"),"HP 87")<>0 THEN
1890         ASSIGN @Hp87xx TO 800
1900         Scode=8
1910         OUTPUT @Hp87xx;"DISP:PROG LOWer"
1920     ELSE
1930         ASSIGN @Hp87xx TO 716
1940         Scode=7
1950     END IF
1960 !
1970 SUBEND !Iden_port
1980 !
```

Transfer of Data to/from the Analyzer

MARKERS	Transferring data using markers. The example also demonstrates the use of the query form of command mnemonics.
⊛ SMITHMKR	Measures reflection of a filter in Smith chart and polar formats.
ASCDATA	Transferring data using the ASCII format.
REALDATA	Transferring data using the IEEE 64-bit floating point REAL format. The example also demonstrates block data transfers of both indefinite length and definite length syntax. Also demonstrated is access to the swapped-byte data format designed for PCs.
INTDATA	Transferring data using the 16-bit INTEGER format.
FAST_CW	Transferring marker data in CW measurement mode.

MARKERS Example Program

This program demonstrates how to transfer measurement data by using the markers. Before any data is read over the HP-IB a controlled sweep should be taken. The analyzer has the ability to process and execute commands very quickly when they are received over the HP-IB. This speed can lead to commands (such as marker searches) being executed before any data has been taken. To ensure that the sweep has completed and the data is present before it is read, the command for a single sweep is used before data is requested. Note that *WAI is sent with that command. More information about making measurements with the analyzer is available in the *User's Guide.*

```
1000 !Filename:  MARKERS
1010 !
1020 ! Description:
1030 !   1. Take sweep
1040 !   2. Set marker to 175 MHz, and query Y value
1050 !   3. Execute Marker -> Max, and query X and Y
1060 !   4. Turn on marker tracking
1070 !   5. Execute a 3 dB bandwidth search
1080 !   6. Query the result
1090 !
1100 COM /Sys_state/ @Hp87xx,Scode
1110 ! Identify I/O Port
1120 CALL Iden_port
1130 !
1140 !
1150 ! Turn on channel 1 and set up start and stop
1160 ! frequencies for the example.  These frequencies
1170 ! were chosen for the demonstration filter that is
1180 ! shipped with the analyzer.
1190 OUTPUT @Hp87xx;"SENS1:STAT ON;FREQ:STAR 10 MHZ;STOP 400 MHZ;*WAI"
1200 !
1210 ! Configure a transmission measurement on channel 1
1220 ! using the narrowband detection mode.
1230 OUTPUT @Hp87xx;"SENS1:FUNC 'XFR:POW:RAT 2,0';DET NBAN;*WAI"
1240 !
1250 ! Take a single controlled sweep and have the
```

```
1260 ! analyzer wait until it has completed before
1270 ! executing the next command.
1280 OUTPUT @Hp87xx;"ABOR;:INIT:CONT OFF;:INIT;*WAI"
1290 !
1300 ! Turn on the first marker.
1310 OUTPUT @Hp87xx;"CALC1:MARK1 ON"
1320 !
1330 ! Set marker 1 to a frequency of 175 MHz.
1340 OUTPUT @Hp87xx;"CALC1:MARK1:X 175 MHZ"
1350 !
1360 ! Query the amplitude of the signal at 175 MHz.
1370 OUTPUT @Hp87xx;"CALC1:MARK1:Y?"
1380 !
1390 ! Read the data; the data is in the NR3 format.
1400 ENTER @Hp87xx;Data_1
1410 DISP "Marker 1 (175 MHz) = ";Data_1
1420 WAIT 5
1430 !
1440 ! Turn on the second marker and use a marker
1450 ! search function to find the maximum point
1460 ! on the data trace.
1470 OUTPUT @Hp87xx;"CALC1:MARK2 ON;MARK2:MAX"
1480 !
1490 ! Query the frequency and amplitude of the
1500 ! maximum point.  Note that the two queries can
1510 ! be combined into one command.
1520 OUTPUT @Hp87xx;"CALC1:MARK2:X?;Y?"
1530 !
1540 ! Read the data.
1550 ENTER @Hp87xx;Freq2,Data2
1560 !
1570 ! Display the results of the marker search.
1580 DISP "Max = ";Data2;"dB at";Freq2/1.E+6;"MHz"
1590 !
1600 ! Put the analyzer into its continuously
1610 ! sweeping mode.  This mode works well for
1620 ! tuning applications.
1630 OUTPUT @Hp87xx;"INIT:CONT ON;*WAI"
1640 !
```

```
1650 ! Turn on the marker search tracking function.
1660 ! This function causes the marker 2 to track
1670 ! the maximum value each time the analyzer takes
1680 ! a sweep.
1690 OUTPUT @Hp87xx;"CALC1:MARK2:FUNC:TRAC ON"
1700 WAIT 5
1710 !
1720 ! Turn off marker 2.
1730 OUTPUT @Hp87xx;"CALC1:MARK2 OFF"
1740 !
1750 ! Take a single controlled sweep.
1760 OUTPUT @Hp87xx;"ABOR;:INIT:CONT OFF;:INIT;*WAI"
1770 !
1780 ! Perform a search for the -3 dB bandwidth of
1790 ! the filter.  This function uses several
1800 ! markers to find four key values.
1810 OUTPUT @Hp87xx;"CALC1:MARK:BWID -3;FUNC:RES?"
1820 !
1830 ! Read the four values:  the bandwidth, center
1840 ! frequency, Q and the insertion loss.
1850 ENTER @Hp87xx;Bwid,Center_f,Q,Loss
1860 !
1870 ! Display the results.
1880 DISP "BW: ";Bwid
1890 WAIT 5
1900 DISP "Center Freq: ";Center_f
1910 WAIT 5
1920 DISP "Q: ";Q
1930 WAIT 5
1940 DISP "Loss: ";Loss
1950 !
1960 ! Turn off all the markers.
1970 OUTPUT @Hp87xx;"CALC1:MARK:AOFF"
1980 END
1990 !
2000 !****************************************************************
2010 ! Iden_port:   Identify io port to use.
2020 ! Description: This routines sets up the I/O port address for
2030 !              the SCPI interface.  For "HP 87xx" instruments,
2040 !              the address assigned to @Hp87xx = 800 otherwise,
2050 !              716.
```

```
2060 !**************************************************************
2070 SUB Iden_port
2080     COM /Sys_state/ @Hp87xx,Scode
2090 !
2100     IF POS(SYSTEM$("SYSTEM ID"),"HP 87")<>0 THEN
2110         ASSIGN @Hp87xx TO 800
2120         Scode=8
2130     ELSE
2140         ASSIGN @Hp87xx TO 716
2150         Scode=7
2160     END IF
2170 !
2180 SUBEND !Iden_port
2190 !
```

⊛SMITHMKR Example Program

```
1000 !Filename:  SMITHCHART
1010 !
1020 !
1030 ! Description:  Measures a 175MHz BPF using the
1040 !   Smith and Polar plot formats.  User must connect
1050 !   the 175MHz filter between the reflection and transmission
1060 !   ports.  The program will do the following:
1070 !     1) Set analyzer to sweep over the filter's passband (50MHz).
1080 !     2) Set analyzer to Smith Chart format; make a marker
1090 !         reading (Frequency, Real Impedance in ohms, Imaginary
              Impedance
1100 !         in ohms, Impedance Capacitance or Inductance); dump the
1110 !         trace and print S11 Real and Imaginary values for the
1120 !         first data point.
1130 !     3) Set analyzer to Polar Chart format; make a marker
1140 !         reading (Frequency, Linear Magnitude in "units",
1150 !         Phase in degrees); dump the
1160 !         trace and print S11 Real and Imaginary values for the
1170 !         first data point.
1180 !
1190 !**************************************************
1200 ! DEFINITIONS
1210 !
1220 REAL Opc,Freq_center,Freq_span,Freq_start,Bpf_q,Bpf_loss
1230 REAL Mrkr_freq,Mrkr_res,Mrkr_reac,Mrkr_ind
1240 REAL Trace_s11(1:201,1:2),Mrkr_mag,Mrkr_phas
1250 !
1260 !***************************************************
1270 ! Determine computer type
1280 !
1290 CLEAR SCREEN
1300 !
1310 !
1320 COM /Sys_state/ @Hp87xx,Scode
1330 ! Identify I/O Port
```

```
1340 CALL Iden_port
1350 !
1360 !
1370 !-------------------------------------------------------
1380 ! Preset analyzer, set Center and Span frequencies
1390 !
1400 OUTPUT @Hp87xx;"SYST:PRES;*OPC?"              !preset instrument
1410 ENTER @Hp87xx;Opc           !waits for PRESET to finish before
     proceeding.
1420 !
1430 ! Center the filter's frequency response (to get an accurate Bandwidth
       measurement).
1440 !
1450 DISP "Setting analyzer frequencies..."          !message to user
1460 OUTPUT @Hp87xx;"ABOR;:INIT:CONT OFF;:INIT;*OPC?" !take a single sweep
1470 ENTER @Hp87xx;Opc                              !wait for sweep to
     finish
1480 OUTPUT @Hp87xx;"CALC1:MARK:FUNC MAX;*WAI"      !set Marker 1 to max
1490 OUTPUT @Hp87xx;"CALC1:MARK:X?;*WAI"            !get Marker frequency
     setting
1500 ENTER @Hp87xx;Mrkr_freq                        !read frequency of max
     marker
1510 OUTPUT @Hp87xx;"SENS1:FREQ:CENT "&VAL$(Mrkr_freq)&" HZ;*WAI"  !set
     Center Freq
1520 OUTPUT @Hp87xx;"SENS1:FREQ:SPAN 200 MHZ;*WAI"  !set Span Freq = 200MHz
1530 !
1540 ! Measure Bandwidth, set Center to band center, Span to 50MHz
1550 !
1560 OUTPUT @Hp87xx;"ABOR;:INIT:CONT OFF;:INIT;*OPC?" !take a single sweep
1570 ENTER @Hp87xx;Opc                              !wait for sweep to
     finish
1580 OUTPUT @Hp87xx;"CALC1:MARK:FUNC BWID;*OPC?"    !search filter for -3dB
     bandwidth
1590 ENTER @Hp87xx;Opc                              !wait for bandwidth to
     be found
1600 OUTPUT @Hp87xx;"CALC1:MARK:FUNC:RES?"          !read the bandwidth data
1610 ENTER @Hp87xx;Freq_span,Freq_center,Bpf_q,Bpf_loss            !read
     in data
1620 OUTPUT @Hp87xx;"SENS1:FREQ:CENT "&VAL$(Freq_center)&" HZ;*WAI"  !set
     Center Freq
```

```
1630 OUTPUT @Hp87xx;"SENS1:FREQ:SPAN 50 MHZ;*WAI"   !set Span Freq to 50MHz
     (passband)
1640 !
1650 !--------------------------------------------------
1660 ! Set marker 1 to beginning of trace.
1670 !
1680 OUTPUT @Hp87xx;"CALC1:MARK:AOFF;*WAI"          !clear all markers
1690 OUTPUT @Hp87xx;"CALC1:MARK1 ON"                !turn on marker 1
1700 OUTPUT @Hp87xx;"SENS1:FREQ:STAR?"              !get start frequency
1710 ENTER @Hp87xx;Freq_start                       !enter start freq
1720 OUTPUT @Hp87xx;"CALC1:MARK1:X "&VAL$(Freq_start)&";*OPC?"  !set marker
     to start freq
1730 ENTER @Hp87xx;Opc                              !wait for all previous
     commands to finish
1740 !
1750 !--------------------------------------------------
1760 ! Set to Reflection mode & Smith Chart format.
1770 !
1780 DISP "Setting to Smith Chart format..."
1790 OUTPUT @Hp87xx;"ABOR;:INIT1:CONT ON;*WAI" !set to Cont Sweep mode so
     can select reflection
1800 OUTPUT @Hp87xx;"SENS1:FUNC 'XFR:POW:RAT 1,0';DET NBAN;*WAI"
         !CHAN1=reflection
1810 OUTPUT @Hp87xx;"CALC1:FORM SMIT;*WAI"          !set smith chart format
1820 !
1830 !--------------------------------------------------
1840 ! Read marker information from Smith Chart.
1850 !
1860 OUTPUT @Hp87xx;"ABOR;:INIT:CONT OFF;:INIT;*OPC?" !force one sweep
     before read markers
1870 ENTER @Hp87xx;Opc                              !wait for sweep to
     finish
1880 OUTPUT @Hp87xx;"ABOR;:INIT1:CONT ON;*WAI"      !set to Continuous Sweep
     mode
1890 OUTPUT @Hp87xx;"CALC1:MARK:X?"                 !read marker frequency
1900 ENTER @Hp87xx;Mrkr_freq                        !units are in Hz
1910 OUTPUT @Hp87xx;"CALC1:MARK:Y:RES?"             !read real part of
     marker impedance
1920 ENTER @Hp87xx;Mrkr_res                         !units are in ohms
1930 OUTPUT @Hp87xx;"CALC1:MARK:Y:REAC?"            !read imaginary part of
     marker impedance
```

```
1940 ENTER @Hp87xx;Mrkr_reac                        !units are in ohms
1950 OUTPUT @Hp87xx;"CALC1:MARK:Y:IND?"             !read inductance (or
     capacitance)
1960 ENTER @Hp87xx;Mrkr_ind    !units are Henries if positive value, Farads
     if negative
1970 !
1980 !------------------------------------------------------------
1990 ! Display Smith Marker data.
2000 !
2010 Mrkr_freq=DROUND(Mrkr_freq,3)                        !round frequency
     to 3 digits
2020 DISP "Smith Marker Frequency = "&VAL$(Mrkr_freq)&"Hz"      !display
     frequency
2030 WAIT 3
2040 !
2050 Mrkr_res=DROUND(Mrkr_res,3)                          !round resistance
     to 3 digits
2060 DISP "Smith Marker Resistance = "&VAL$(Mrkr_res)&" ohms"
2070 WAIT 3
2080 !
2090 Mrkr_reac=DROUND(Mrkr_reac,3)                        !round reactance
     to 3 digits
2100 DISP "Smith Marker Reactance = "&VAL$(Mrkr_reac)&" ohms"
2110 WAIT 3
2120 !
2130 Mrkr_ind=DROUND(Mrkr_ind,3)                          !round inductance
     to 3 digits
2140 IF Mrkr_ind<0 THEN                                   !label as
     capacitance if negative
2150    DISP "Smith Marker Capacitance = "&VAL$(-Mrkr_ind)&"F"!label
        capacitance
2160 ELSE                                                 !label as
     inductance if positive
2170    DISP "Smith Marker Inductance = "&VAL$(Mrkr_ind)&"H"  !label
        inductance
2180 END IF
2190 WAIT 3
2200 !
```

```
2210 !----------------------------------------------------
2220 ! Read Smith Chart formatted trace data, display first data point.
2230 !   Data is transferred in ASCII format with 3 significant digits.
2240 !   S11 trace data is read out as:  Real data for point #1, Imaginary
        data
2250 !   for point #1, Real data for point #2, Imaginary data for point
        #2...
2260 !   Since instrument was preset, number of trace data points
2270 !   defaults to 201.
2280 !
2290 OUTPUT @Hp87xx;"FORM:DATA ASC,3;:TRAC? CH1FDATA"      !set up to read
     ASCII data, 3 digits
2300 ENTER @Hp87xx;Trace_s11(*)                           !read trace data,
     real & imaginary pairs
2310 !
2320 ! Display data.
2330 !
2340 DISP "Smith Trace Point #1:  S11(REAL) = "&VAL$(Trace_s11(1,1))&"
     Units"  !display Real data
2350 WAIT 3
2360 DISP "Smith Trace Point #1:  S11(IMAGINARY) = "&VAL$(Trace_s11(1,2))&"
     Units"  !display Imaginary data
2370 WAIT 3
2380 !
2390 !+++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
2400 ! Set to Polar Chart Format, read Polar Markers.
2410 !
2420 DISP "Setting to Polar Format..."
2430 OUTPUT @Hp87xx;"CALC1:FORM POL;*WAI"           !set polar chart format
2440 OUTPUT @Hp87xx;"CALC1:MARK:X?"                 !read marker frequency
2450 ENTER @Hp87xx;Mrkr_freq                        !units are in Hz
2460 OUTPUT @Hp87xx;"CALC1:MARK:Y:MAGN?"            !read magnitude marker
     reflection coefficient
2470 ENTER @Hp87xx;Mrkr_mag                         !magnitude in "units"
2480 OUTPUT @Hp87xx;"CALC1:MARK:Y:PHAS?"            !read phase of marker
     reflection coefficient
2490 ENTER @Hp87xx;Mrkr_phas                        !units are in degrees
2500 !
```

```
2510 !-----------------------------------------------------------
2520 ! Display Polar Marker data.
2530 !
2540 Mrkr_freq=DROUND(Mrkr_freq,3)                          !round frequency
     to 3 digits
2550 DISP "Polar Marker Frequency = "&VAL$(Mrkr_freq)&"Hz"       !display
     frequency
2560 WAIT 3
2570 !
2580 Mrkr_mag=DROUND(Mrkr_mag,3)                            !round magnitude
     to 3 digits
2590 DISP "Polar Marker Magnitude = "&VAL$(Mrkr_mag)&" Units"    !display
     magnitude
2600 WAIT 3
2610 !
2620 Mrkr_phas=DROUND(Mrkr_phas,3)                          !round phase to 3
     digits
2630 DISP "Polar Marker Phase = "&VAL$(Mrkr_phas)&" Degrees"     !display
     phase
2640 WAIT 3
2650 !
2660 !----------------------------------------------------
2670 ! Read Polar Chart trace data, display first data point.
2680 !   S11 trace data is read out as:  Real data for point #1, Imaginary
         data
2690 !   for point #1, Real data for point #2, Imaginary data for point
         #2...
2700 !
2710 OUTPUT @Hp87xx;"FORM:DATA ASC,3;:TRAC? CH1FDATA"      !set up to read
     ASCII data, 3 digits
2720 ENTER @Hp87xx;Trace_s11(*)                            !read trace data,
     real & imaginary pairs
2730 !
2740 ! Display data
2750 !
2760 DISP "Polar Trace Point #1:  S11(REAL) = "&VAL$(Trace_s11(1,1))&"
     Units"  !display Real data
2770 WAIT 3
2780 DISP "Polar Trace Point #1:  S11(IMAGINARY) = "&VAL$(Trace_s11(1,2))&"
     Units"  !display Imaginary data
2790 WAIT 3
```

```
2800 DISP ""                                                 !clear display
     line
2810 !
2820 STOP
2830 END
2840 !
2850 !**********************************************************
2860 ! Iden_port:   Identify io port to use.
2870 ! Description: This routines sets up the I/O port address for
2880 !              the SCPI interface.  For "HP 87xx" instruments,
2890 !              the address assigned to @Hp87xx = 800 otherwise,
2900 !              716.
2910 !**********************************************************
2920 SUB Iden_port
2930     COM /Sys_state/ @Hp87xx,Scode
2940 !
2950     IF POS(SYSTEM$("SYSTEM ID"),"HP 87")<>0 THEN
2960         ASSIGN @Hp87xx TO 800
2970         Scode=8
2980     ELSE
2990         ASSIGN @Hp87xx TO 716
3000         Scode=7
3010     END IF
3020 !
3030 SUBEND !Iden_port
3040 !
```

ASCDATA Example Program

This program demonstrates how to read data arrays from the analyzer and write them back again. The ASCii data format is being used with a resolution of 5 digits. More information about data transfer is available in Chapter 4, "Data Types and Encoding," and Chapter 6, "Trace Data Transfers."

In addition to the channel 1 formatted data array used in this example, there are a number of arrays that can be accessed inside the instrument. These arrays and their corresponding mnemonics are listed in Chapter 6 in Table 6-4 and Table 6-5.

```
1000 !Filename:  ASCDATA
1010 !
1020 ! Description:
1030 !  1. Takes a sweep, and reads the formatted
1040 !     data trace into an array.  The trace
1050 !     is read as a definite length block.
1060 !  2. Instructs you to remove DUT.
1070 !  3. Downloads the trace back to the analyzer
1080 !     as an indefinite length block.
1090 REAL Data1(1:51)
1100 !
1110 COM /Sys_state/ @Hp87xx,Scode
1120 ! Identify I/O Port
1130 CALL Iden_port
1140 !
1150 !
1160 ! Set the analyzer to measure 51 data points.
1170 OUTPUT @Hp87xx;"SENS1:SWE:POIN 51;*WAI"
1180 !
1190 ! Take a single sweep, leaving the analyzer
1200 ! in trigger hold mode.
1210 OUTPUT @Hp87xx;"ABOR;:INIT1:CONT OFF;:INIT1;*WAI"
1220 !
1230 ! Set up the ASCII data format with 5
1240 ! significant digits
1250 OUTPUT @Hp87xx;"FORM:DATA ASC,5"
```

```
1260 !
1270 ! request the channel 1 formatted data array
1280 ! from the instrument.
1290 OUTPUT @Hp87xx;"TRAC? CH1FDATA"
1300 !
1310 ! Get the data and put into data array Data1.
1320 ENTER @Hp87xx;Data1(*)
1330 !
1340 ! Display the first 3 numbers in the array.
1350 DISP "Trace: ";Data1(1);Data1(2);Data1(3);"..."
1360 !
1370 ! Use the wait time to visually compare the
1380 ! numbers to the visible data trace.
1390 WAIT 5
1400 !
1410 ! Prompt the operator to disconnect the test
1420 ! device and then how to continue the program.
1430 DISP "Disconnect the test device -- Press Continue"
1440 PAUSE
1450 !
1460 ! Update the display line.
1470 DISP "Taking a new sweep...";
1480 !
1490 ! Take a sweep so the display shows new data.
1500 OUTPUT @Hp87xx;":INIT1;*WAI"
1510 DISP " Done."
1520 WAIT 5
1530 !
1540 ! Prepare the analyzer to receive the data.
1550 ! Suppress the "end" character by using a
1560 ! semicolon at end of output statement.
1570 DISP "Downloading saved trace...";
1580 OUTPUT @Hp87xx;"TRAC CH1FDATA";
1590 !
1600 ! Send the data array one point at a time,
1610 ! using the semicolon at the end of the
1620 ! output statement to suppress the
1630 ! end character.
1640 FOR I=1 TO 51
1650     OUTPUT @Hp87xx;", ";Data1(I);
1660 NEXT I
```

```
1670 !
1680 ! Now send the end character.
1690 OUTPUT @Hp87xx;""
1700 DISP " Done!"
1710 END
1720 !
1730 !**********************************************************
1740 ! Iden_port:   Identify io port to use.
1750 ! Description: This routines sets up the I/O port address for
1760 !              the SCPI interface.  For "HP 87xx" instruments,
1770 !              the address assigned to @Hp87xx = 800 otherwise,
1780 !              716.
1790 !**********************************************************
1800 SUB Iden_port
1810     COM /Sys_state/ @Hp87xx,Scode
1820 !
1830     IF POS(SYSTEM$("SYSTEM ID"),"HP 87")<>0 THEN
1840         ASSIGN @Hp87xx TO 800
1850         Scode=8
1860     ELSE
1870         ASSIGN @Hp87xx TO 716
1880         Scode=7
1890     END IF
1900 !
1910 SUBEND !Iden_port
1920 !
```

REALDATA Example Program

This program demonstrates how to read data arrays from the analyzer and write them back again. The `REAL,64` data format is being used. Note that the analyzer outputs the data using the definite length block syntax. This example uses the indefinite length block syntax when data is being written back to the analyzer. More information about data transfer is available in Chapter 4, "Data Types and Encoding." All of the arrays listed in the `ASCDATA` example section can also be accessed using this data format.

```
1000 !Filename:  REALDATA
1010 !
1020 ! Description:
1030 !  1. Takes a sweep, and reads the formatted
1040 !     data trace into an array.  The trace
1050 !     is read as a definite length block.
1060 !  2. Instructs you to remove DUT.
1070 !  3. Downloads the trace back to the analyzer
1080 !     as an indefinite length block.
1090 DIM A$[10],Data1(1:51)
1100 INTEGER Digits,Bytes
1110 !
1120 COM /Sys_state/ @Hp87xx,Scode
1130 ! Identify I/O Port
1140 CALL Iden_port
1150 !
1160 !
1170 ! Set up the analyzer to measure 51 data points.
1180 OUTPUT @Hp87xx;"SENS1:SWE:POIN 51;*WAI"
1190 !
1200 ! Take a single sweep, leaving the analyzer
1210 ! in trigger hold mode.
1220 OUTPUT @Hp87xx;"ABOR;:INIT1:CONT OFF;:INIT1;*WAI"
1230 !
1240 ! Select binary block transfer.
1250 OUTPUT @Hp87xx;"FORM:DATA REAL,64"
1260 !
1270 ! Request the channel 1 formatted data array
1280 ! from the analyzer.
```

```
1290 OUTPUT @Hp87xx;"TRAC? CH1FDATA"
1300 !
1310 ! Turn on ASCII formatting on the I/O path.
1320 ! It is needed for reading the header
1330 ! information.
1340 ASSIGN @Hp87xx;FORMAT ON
1350 !
1360 ! Get the data header.  "A$" will contain the
1370 ! "#" character indicating a block data transfer.
1380 ! "Digits" will contain the number of characters
1390 ! for the number of bytes value which follows.
1400 ENTER @Hp87xx USING "%,A,D";A$,Digits
1410 !
1420 ! Get the rest of the header.  The number of
1430 ! bytes to capture in the data array will be
1440 ! placed in "Bytes".  Note the use of "Digits"
1450 ! in the IMAGE string.
1460 ENTER @Hp87xx USING "%,"&VAL$(Digits)&"D";Bytes
1470 !
1480 ! Turn off ASCII formatting on the I/O path;
1490 ! it is not needed for transferring binary
1500 ! formatted data.
1510 ASSIGN @Hp87xx;FORMAT OFF
1520 !
1530 ! Get the data.
1540 ENTER @Hp87xx;Data1(*)
1550 !
1560 ! Turn on ASCII formatting again.
1570 ASSIGN @Hp87xx;FORMAT ON
1580 !
1590 ! Get the "end of data" character.
1600 ENTER @Hp87xx;A$
1610 !
1620 ! Display the first three numbers in the array.
1630 DISP "Trace: ";Data1(1);Data1(2);Data1(3);"..."
1640 !
1650 ! Use this time to visually compare the
1660 ! numbers to the visible data trace.
1670 WAIT 5
1680 !
1690 ! Prompt the operator to disconnect the test
```

```
1700 ! device and how to continue the program.
1710 DISP "Disconnect the test device -- Press Continue"
1720 PAUSE
1730 !
1740 ! Update the display line.
1750 DISP "Taking a new sweep...";
1760 !
1770 ! Take a sweep so the display shows new data.
1780 OUTPUT @Hp87xx;":INIT1;*WAI"
1790 DISP " Done."
1800 WAIT 5
1810 !
1820 ! Send the header for an indefinite block length
1830 ! data transfer.
1840 DISP "Downloading saved trace...";
1850 OUTPUT @Hp87xx;"TRAC CH1FDATA, #0";
1860 !
1870 ! Turn off ASCII formatting.
1880 ASSIGN @Hp87xx;FORMAT OFF
1890 !
1900 ! Send the data array back to the analyzer.
1910 OUTPUT @Hp87xx;Data1(*),END
1920 !
1930 ! Turn on ASCII formatting again.
1940 ASSIGN @Hp87xx;FORMAT ON
1950 DISP " Done!"
1960 END
1970 !
1980 !************************************************************
1990 ! Iden_port:   Identify io port to use.
2000 ! Description: This routines sets up the I/O port address for
2010 !              the SCPI interface.  For "HP 87xx" instruments,
2020 !              the address assigned to @Hp87xx = 800 otherwise,
2030 !              716.
2040 !************************************************************
```

```
2050 SUB Iden_port
2060     COM /Sys_state/ @Hp87xx,Scode
2070 !
2080     IF POS(SYSTEM$("SYSTEM ID"),"HP 87")<>0 THEN
2090         ASSIGN @Hp87xx TO 800
2100         Scode=8
2110     ELSE
2120         ASSIGN @Hp87xx TO 716
2130         Scode=7
2140     END IF
2150 !
2160 SUBEND !Iden_port
2170 !
```

INTDATA Example Program

This program demonstrates how to read data arrays from the analyzer and write them back again. The `INTeger,16` data format is being used. This data format is the instrument's internal format. It should only be used to read data that will later be returned to the instrument.

The data array dimensioned in line `1100` is different from the arrays in either `REAL,64` or `ASCii` examples. This is because each data point is represented by a set of three 16-bit integers. Another difference in using this data format is that all arrays cannot be accessed with it. The formatted data arrays `CH1FDATA` and `CH2FDATA` cannot be read using the `INTEGER` format.

Note that the analyzer outputs the data using the definite length block syntax. This example uses the indefinite length block syntax when data is being written back to the analyzer. More information about data transfer is available in Chapter 4, "Data Types and Encoding."

```
1000 !Filename:  INTDATA
1010 !
1020 ! Description:
1030 !  1. Takes a sweep, and reads the formatted
1040 !     data trace into an array.  The trace
1050 !     is read as a definite length block.
1060 !  2. Instructs you to remove DUT.
1070 !  3. Downloads the trace back to the analyzer
1080 !     as an indefinite length block.
1090 DIM A$[10]
1100 INTEGER Digits,Bytes,Data1(1:51,1:3)
1110 !
1120 COM /Sys_state/ @Hp87xx,Scode
1130 ! Identify I/O Port
1140 CALL Iden_port
1150 !
1160 !
1170 ! Set up the analyzer to measure 51 data points.
1180 OUTPUT @Hp87xx;"SENS1:SWE:POIN 51;*WAI"
1190 !
1200 ! Take a single sweep, leaving the analyzer
1210 ! in trigger hold mode.
```

```
1220 OUTPUT @Hp87xx;"ABOR;:INIT1:CONT OFF;:INIT1;*WAI"
1230 !
1240 ! Select binary block transfer
1250 OUTPUT @Hp87xx;"FORM:DATA INT,16"
1260 !
1270 ! Request the channel 1 unformatted data array
1280 ! from the analyzer.
1290 OUTPUT @Hp87xx;"TRAC? CH1SDATA"
1300 !
1310 ! Turn on ASCII formatting on the I/O path;
1320 ! it is needed for reading the header information.
1330 ASSIGN @Hp87xx;FORMAT ON
1340 !
1350 ! Get the data header.  "A$" will contain the
1360 ! "#" character indicating a block data transfer.
1370 ! "Digits" will contain the number of characters
1380 ! for the number of bytes value which follows.
1390 ENTER @Hp87xx USING "%,A,D";A$,Digits
1400 !
1410 ! Get the rest of the header.  The number of
1420 ! bytes to capture in the data array will be
1430 ! placed in "Bytes".  Note the use of "Digits"
1440 ! in the IMAGE string.
1450 ENTER @Hp87xx USING "%,"&VAL$(Digits)&"D";Bytes
1460 !
1470 ! Turn off ASCII formatting on the I/O path;
1480 ! it is not needed for transferring binary
1490 ! formatted data.
1500 ASSIGN @Hp87xx;FORMAT OFF
1510 !
1520 ! Get the data.
1530 ENTER @Hp87xx;Data1(*)
1540 !
1550 ! Turn on ASCII formatting again.
1560 ASSIGN @Hp87xx;FORMAT ON
1570 !
1580 ! Get the "end of data" character.
1590 ENTER @Hp87xx;A$
1600 !
1610 ! Display the first 3 numbers; there will
1620 ! be no visible similarity between these
```

```
1630 ! numbers and the data displayed on the
1640 ! analyzer.
1650 DISP "Trace: ";Data1(1,1);Data1(1,2);Data1(1,3);"..."
1660 WAIT 5
1670 !
1680 ! Prompt the operator to disconnect the test
1690 ! device and how to continue the program.
1700 DISP "Disconnect the test device -- Press Continue"
1710 PAUSE
1720 !
1730 ! Update the display line.
1740 DISP "Taking a new sweep...";
1750 !
1760 ! Take a sweep so the display shows new data.
1770 OUTPUT @Hp87xx;":INIT1;*WAI"
1780 DISP " Done."
1790 WAIT 5
1800 !
1810 ! Send the header for an indefinite block length
1820 ! data transfer.
1830 DISP "Downloading saved trace...";
1840 OUTPUT @Hp87xx;"TRAC CH1SDATA, #0";
1850 !
1860 ! Turn off ASCII formatting.
1870 ASSIGN @Hp87xx;FORMAT OFF
1880 !
1890 ! Send the data back to the analyzer.
1900 OUTPUT @Hp87xx;Data1(*),END
1910 !
1920 ! Turn on ASCII formatting.
1930 ASSIGN @Hp87xx;FORMAT ON
1940 DISP "Done!"
1950 END
1960 !
1970 !************************************************************
1980 ! Iden_port:   Identify io port to use.
1990 ! Description: This routines sets up the I/O port address for
2000 !              the SCPI interface.  For "HP 87xx" instruments,
2010 !              the address assigned to @Hp87xx = 800 otherwise,
2020 !              716.
2030 !************************************************************
```

```
2040 SUB Iden_port
2050     COM /Sys_state/ @Hp87xx,Scode
2060 !
2070     IF POS(SYSTEM$("SYSTEM ID"),"HP 87")<>0 THEN
2080         ASSIGN @Hp87xx TO 800
2090         Scode=8
2100     ELSE
2110         ASSIGN @Hp87xx TO 716
2120         Scode=7
2130     END IF
2140 !
2150 SUBEND !Iden_port
2160 !
```

FAST_CW Example Program

This program demonstrates how to set up a CW (fixed frequency) sweep with the minimum number of trace points. Such a sweep allows measurements to made very rapidly. The program also shows how to set up a loop which uses a fast CW sweep, reads a marker value on the measurement trace, then changes the CW frequency.

```
1000 ! Filename: FAST_CW
1010 !
1020 ! Description:
1030 !   Set sweep to CW, and select the
1040 !   fewest number of points.
1050 !   Change the frequency, take a sweep,
1060 !   and use a marker to read the trace.
1070 !   Repeat as quickly as possible.
1080 !
1090 DIM Freq_str$[20]
1100 DIM Msg$[100]
1110 !
1120 !
1130 COM /Sys_state/ @Hp87xx,Scode
1140 ! Identify I/O Port.
1150 CALL Iden_port
1160 !
1170 !
1180 ! PRESET, to ensure known state.
1190 OUTPUT @Hp87xx;"SYST:PRES;*WAI"
1200 CLEAR SCREEN
1210 !
1220 ! Set up the analyzer to measure 3 data points.
1230 OUTPUT @Hp87xx;"SENS1:SWE:POIN 3;*WAI"
1240 !
1250 ! Select CW display and sweep.
1260 OUTPUT @Hp87xx;"DISP:ANN:FREQ1:MODE CW"
1270 OUTPUT @Hp87xx;"SENS1:FREQ:SPAN 0 Hz;*WAI"
1280 !
1290 ! Take a single sweep, leaving the analyzer
1300 ! in trigger hold mode.
```

```
1310 OUTPUT @Hp87xx;"ABOR;:INIT1:CONT OFF;*WAI"
1320 !
1330 ! Turn on Marker 1
1340 OUTPUT @Hp87xx;"CALC:MARK1 ON"
1350 !
1360 Count=0
1370 T0=TIMEDATE
1380 ! Step from 175 MHz 463 MHz by 6 MHz
1390 FOR Freq=175 TO 463 STEP 6
1400   ! Take a sweep
1410     Freq_str$=VAL$(Freq)&" MHz"
1420     OUTPUT @Hp87xx;"SENS1:FREQ:CENT ";Freq_str$
1430     OUTPUT @Hp87xx;"INIT1;*WAI"
1440   !
1450   ! Set marker to frequency
1460     OUTPUT @Hp87xx;"CALC:MARK:X ";Freq_str$
1470   !
1480   ! Query the marker value
1490     OUTPUT @Hp87xx;"CALC:MARK:Y?"
1500     ENTER @Hp87xx;Response
1510   !
1520   ! Display the first three numbers in the array.
1530     Msg$="'"&Freq_str$&": "&VAL$(Response)&"'"
1540     OUTPUT @Hp87xx;"DISP:ANN:MESS ";Msg$
1550     PRINT Msg$
1560     Count=Count+1
1570 NEXT Freq
1580 T1=TIMEDATE
1590 PRINT "Sweeps per second: ";Count/(T1-T0)
1600 DISP "Sweeps per second: ";Count/(T1-T0)
1610 END
1620 !
```

```
1630 !*************************************************************
1640 ! Iden_port:   Identify io port to use.
1650 ! Description: This routines sets up the I/O port address for
1660 !              the SCPI interface.  For "HP 87xx" instruments,
1670 !              the address assigned to @Hp87xx = 800 otherwise,
1680 !              716.
1690 !*************************************************************
1700 SUB Iden_port
1710     COM /Sys_state/ @Hp87xx,Scode
1720 !
1730     IF POS(SYSTEM$("SYSTEM ID"),"HP 87")<>0 THEN
1740        ASSIGN @Hp87xx TO 800
1750        Scode=8
1760     ELSE
1770        ASSIGN @Hp87xx TO 716
1780        Scode=7
1790     END IF
1800 !
1810 SUBEND !Iden_port
1820 !
```

Calibration

TRANCAL	Performing a transmission calibration. The calibration is User Defined (performed over the instruments current source settings). This example also demonstrates the use of the `*OPC?` command.
REFLCAL	Performing a reflection calibration. The calibration is Full Band (performed over the instrument's preset source settings). This example also demonstrates the detection of front panel key presses, the use of softkeys, and the use of the `*OPC?` command.
LOADCALS	Uploading and downloading correction arrays. The data transfer is performed in the 16-bit integer format. The arrays must be dimensioned properly for both the number of data points and the format of the data being transferred.
CALKIT	Instrument state file for downloading User Defined cal kit definitions. This example is NOT a program. It is an instrument state file example. This type of file enables the user to calibrate the analyzer for use with connector types that are not in the firmware. See "Writing and Editing Your Own Cal Kit File" in Chapter 6 of the *User's Guide*.

TRANCAL Example Program

This program demonstrates a transmission calibration performed over user-defined source settings (frequency range, power and number of points). The operation complete query is used at each step in the process to make sure the steps are taken in the correct order. More information on calibration is available in the *User's Guide.*

```
1000 ! Filename:  TRANCAL
1010 !
1020 ! Guide user through a transmission cal.
1030 !
1040 !
1050 COM /Sys_state/ @Hp87xx,Scode
1060 ! Identify I/O Port
1070 CALL Iden_port
1080 !
1090 !
1100 ! Configure the analyzer to measure transmission
1110 ! on channel 1.
1120 OUTPUT @Hp87xx;"SENS1:FUNC 'XFR:POW:RAT 2,0';DET NBAN;*WAI"
1130 !
1140 ! Select a calibration kit type.
1150 OUTPUT @Hp87xx;"SENS:CORR:COLL:CKIT 'COAX,7MM,TYPE-N,50,FEMALE'"
1160 !
1170 ! Select a transmission calibration for the current
1180 ! analyzer settings.  The "IST:OFF" ensures that
1190 ! the current settings will be used.
1200 OUTPUT @Hp87xx;"SENS1:CORR:COLL:IST OFF;METH TRAN1"
1210 !
1220 ! Prompt the operator to make a through
1230 ! connection.
1240 DISP "Connect THRU - Press Continue"
1250 PAUSE
1260 DISP "Measuring THRU"
1270 !
1280 ! Analyzer measures the through.
1290 OUTPUT @Hp87xx;"SENS1:CORR:COLL STAN1;*OPC?"
1300 !
```

```
1310 ! Wait until the measurement is complete.
1320 ENTER @Hp87xx;Opc
1330 DISP "Calculating Error Coefficients"
1340 !
1350 ! Tell the analyzer to calculate the
1360 ! error coefficients after the measurement
1370 ! is made, and then save for use during
1380 ! subsequent transmission measurements.
1390 ! Note that this is not the same as using
1400 ! the SAVE RECALL key functionality.
1410 OUTPUT @Hp87xx;"SENS1:CORR:COLL:SAVE;*OPC?"
1420 !
1430 ! Wait for the calculations and save to be
1440 ! completed.
1450 ENTER @Hp87xx;Opc
1460 DISP "User Defined TRANSMISSION CAL COMPLETED!"
1470 END
1480 !
1490 !****************************************************************
1500 ! Iden_port:   Identify io port to use.
1510 ! Description: This routines sets up the I/O port address for
1520 !              the SCPI interface.  For "HP 87xx" instruments,
1530 !              the address assigned to @Hp87xx = 800 otherwise,
1540 !              716.
1550 !****************************************************************
1560 SUB Iden_port
1570     COM /Sys_state/ @Hp87xx,Scode
1580 !
1590     IF POS(SYSTEM$("SYSTEM ID"),"HP 87")<>0 THEN
1600         ASSIGN @Hp87xx TO 800
1610         Scode=8
1620     ELSE
1630         ASSIGN @Hp87xx TO 716
1640         Scode=7
1650     END IF
1660 !
1670 SUBEND !Iden_port
1680 !
```

REFLCAL Example Program

This program demonstrates a reflection calibration performed over the preset source settings (frequency range, power and number of points). The operation complete query is used at each step in the process to make sure the steps are taken in the correct order. More information on calibration is available in the *User's Guide.*

```
1000 !Filename:  REFLCAL
1010 !
1020 ! Guide user through a reflection cal.
1030 !
1040 DIM Msg$[50]
1050 !
1060 COM /Sys_state/ @Hp87xx,Scode,Internal
1070 ! Identify I/O Port
1080 CALL Iden_port
1090 !
1100 !
1110 ! Configure the analyzer to measure
1120 ! reflection on channel 1.
1130 OUTPUT @Hp87xx;"SENS1:FUNC 'XFR:POW:RAT 1,0';DET NBAN;*WAI"
1140 !
1150 ! Select Calibration Kit for 50 ohm instruments.
1160 OUTPUT @Hp87xx;"SENS:CORR:COLL:CKIT 'COAX,7MM,TYPE-N,50,FEMALE'"
1170 !
1180 ! Select Calibration Kit for 75 ohm instruments.
1190 ! (Comment out the 50 ohm line above and uncomment the line
1200 ! below.)
1210 ! OUTPUT @Hp87xx;"SENS:CORR:COLL:CKIT 'COAX,7MM,TYPE-N,75,FEMALE'"
1220 !
1230 ! Select a reflection calibration for the current
1240 ! analyzer settings.  The "IST:OFF" ensures that
1250 ! current settings will be used.
1260 OUTPUT @Hp87xx;"SENS1:CORR:COLL:IST OFF;METH REFL3"
1270 !
1280 ! Prompt the operator to connect an open.
1290 Msg$="Connect OPEN"
1300 GOSUB Get_continue
```

```
1310 DISP "Measuring OPEN"
1320 !
1330 ! Measure the open.
1340 OUTPUT @Hp87xx;"SENS1:CORR:COLL STAN1;*OPC?"
1350 !
1360 ! Wait until the measurement of the open
1370 ! is complete.
1380 ENTER @Hp87xx;Opc
1390 !
1400 ! Prompt the operator to connect a short.
1410 Msg$="Connect SHORT"
1420 GOSUB Get_continue
1430 DISP "Measuring SHORT"
1440 !
1450 ! Measure the short.
1460 OUTPUT @Hp87xx;"SENS1:CORR:COLL STAN2;*OPC?"
1470 !
1480 ! Wait until measurement of the short
1490 ! is complete.
1500 ENTER @Hp87xx;Opc
1510 !
1520 ! Prompt operator to connect a load.
1530 Msg$="Connect LOAD"
1540 GOSUB Get_continue
1550 DISP "Measuring LOAD"
1560 !
1570 ! Measure the load.
1580 OUTPUT @Hp87xx;"SENS1:CORR:COLL STAN3;*OPC?"
1590 ! Wait until measurement of the load
1600 ! is complete.
1610 ENTER @Hp87xx;Opc
1620 DISP "Calculating Error Coefficients"
1630 !
1640 ! Tell the analyzer to calculate the
1650 ! error coefficients, and then save
1660 ! for use during subsequent reflection
1670 ! measurements. Note that this is not
1680 ! the same as using the SAVE RECALL key
1690 ! functionality.
1700 OUTPUT @Hp87xx;"SENS1:CORR:COLL:SAVE;*OPC?"
1710 !
```

```
1720 ! Wait for the calculations to be completed
1730 ! and the calibration saved.
1740 ENTER @Hp87xx;Opc
1750 DISP "Full Band REFLECTION CAL COMPLETED!"
1760 STOP
1770 !
1780 Get_continue: ! Subroutine to handle operator prompts.
1790 !
1800 ! "Internal" is determined above based on the
1810 ! controller.
1820 IF Internal=1 THEN
1830 !
1840 ! If internal control, then use the display
1850 ! line for the prompt.
1860     DISP Msg$&" - Press Measure Standard"
1870 !
1880 ! Use the softkey 2 for the response; loop
1890 ! while waiting for it to be pressed.
1900     ON KEY 2 LABEL "Measure    Standard" RECOVER Go_on
1910     LOOP
1920     END LOOP
1930 ELSE
1940 !
1950 ! If external control, clear the key queue
1960 ! so previous key presses will not interfere.
1970     OUTPUT @Hp87xx;"SYST:KEY:QUE:CLE"
1980 !
1990 ! Use the BEGIN key for the response.
2000     DISP Msg$&" - Press BEGIN to continue"
2010 !
2020 ! Turn on the key queue to trap all key
2030 ! presses.
2040     OUTPUT @Hp87xx;"SYST:KEY:QUE ON"
2050 !
2060 ! Loop while waiting for a key to be
2070 ! pressed.
2080     LOOP
2090 ! Query the device status condition
2100 ! register.
2110         OUTPUT @Hp87xx;"STAT:DEV:COND?"
2120         ENTER @Hp87xx;Dev_cond
```

```
2130 !
2140 ! Check the bit that indicates a key press.
2150         IF BIT(Dev_cond,0)=1 THEN
2160             OUTPUT @Hp87xx;"SYST:KEY?"
2170             ENTER @Hp87xx;Key_code
2180         END IF
2190 !
2200 ! Stop looping if the BEGIN key was pressed.
2210     EXIT IF Key_code=40
2220     END LOOP
2230     Key_code=0
2240 END IF
2250 !
2260 Go_on: ! Subroutine to turn off the softkeys
2270 ! on the analyzer and the computer,
2280 ! and return to main body of the
2290 ! program.
2300 OFF KEY
2310 RETURN
2320 END
2330 !
2340 !**************************************************************
2350 ! Iden_port:   Identify io port to use.
2360 ! Description: This routines sets up the I/O port address for
2370 !              the SCPI interface.  For "HP 87xx" instruments,
2380 !              the address assigned to @Hp87xx = 800 otherwise,
2390 !              716.
2400 !**************************************************************
```

```
2410 SUB Iden_port
2420     COM /Sys_state/ @Hp87xx,Scode,Internal
2430 !
2440     IF POS(SYSTEM$("SYSTEM ID"),"HP 87")<>0 THEN
2450         ASSIGN @Hp87xx TO 800
2460         Scode=8
2470         Internal=1
2480     ELSE
2490         ASSIGN @Hp87xx TO 716
2500         Scode=7
2510         Internal=0
2520     END IF
2530 !
2540 SUBEND !Iden_port
2550 !
```

LOADCALS Example Program

This program demonstrates how to read the correction arrays from the analyzer and write them back again. The `INTeger,16` data format is being used because the data does not need to be interpreted, only stored and retrieved. More information about calibration is available in the *User's Guide*.

The size of the arrays into which the data is read is critical. If they are not dimensioned correctly the program will not work. Most correction arrays, including the factory default (`DEF`) and the full band (`FULL`, preset source settings) arrays have 801 points. For user defined calibrations (`USER`) the number of points must be determined. If the number of points is other than 801, lines `1110` and `1790` will need to be changed to allocate arrays for the correct number of points. The number of points can be found by reading the correction array's header and determining the size as shown in the example below.

```
1000 !Filename:  LOADCALS
1010 !
1020 ! Description:
1030 !  1. Query the calibration arrays, based on
1040 !     the current measurement (trans/refl).
1050 !  2. Change number of points to 801.
1060 !  3. Download the calibration arrays back
1070 !     into the analyzer.
1080 !
1090 DIM Func$[20],A$[10]
1100 INTEGER Swap,Arrays,Digits,Bytes,Points
1110 INTEGER Corr1(1:801,1:3),Corr2(1:801,1:3),Corr3(1:801,1:3)
1120 !
1130 COM /Sys_state/ @Hp87xx,Scode
1140 ! Identify I/O Port
1150 CALL Iden_port
1160 !
1170 !
1180 ! Query the measurement parameter.
1190 OUTPUT @Hp87xx;"SENS1:FUNC?"
1200 !
1210 ! Read the analyzer's response.
1220 ENTER @Hp87xx;Func$
1230 !
1240 ! Set up a SELECT/CASE depending on the
1250 ! response.
1260 SELECT Func$
1270 !
1280 ! This is the transmission case, a ratio of
1290 ! the powers measured by detector 2 (B) and
1300 ! detector 0 (R).
1310 CASE """XFR:POW:RAT 2, 0"""
1320 !
1330 ! The transmission calibration has only one
1340 ! correction array.
1350    Arrays=1
1360 !
1370 ! This is the reflection case, a ratio of
1380 ! the powers measured by detector 1 (A) and
1390 ! detector 0 (R).
1400 CASE """XFR:POW:RAT 1, 0"""
```

```
1410 !
1420 ! The reflection calibration has 3 correction
1430 ! arrays.
1440     Arrays=3
1450 END SELECT
1460 !
1470 ! Select the 16 bit integer binary data format.
1480 OUTPUT @Hp87xx;"FORM:DATA INT,16"
1490 !
1500 ! Select normal byte order.
1510 OUTPUT @Hp87xx;"FORM:BORD NORM"
1520 !
1530 ! Request the first correction array from the a
1540 ! analyzer.
1550 OUTPUT @Hp87xx;"TRAC? CH1SCORR1"
1560 !
1570 ! Turn on ASCII formatting on the I/O path
1580 ! to read the header information.
1590 ASSIGN @Hp87xx;FORMAT ON
1600 !
1610 ! Get the header, including the number of
1620 ! of characters that will hold the number
1630 ! of bytes value which follows.
1640 ENTER @Hp87xx USING "%,A,D";A$,Digits
1650 !
1660 ! Get the rest of the header.  The number
1670 ! of bytes to capture in the correction
1680 ! array will be placed in "Bytes".  Note
1690 ! the use of "Digits" in the IMAGE string.
1700 ENTER @Hp87xx USING "%,"&VAL$(Digits)&"D";Bytes
1710 !
1720 ! Determine the number of points from the
1730 ! number of bytes (6 bytes per point).
1740 Points=Bytes/6
1750 !
1760 ! This example was set up in line 1110 above
1770 ! for 801 points.  Edit this line and line 1110
1780 ! to allow other dimensions.
1790 IF Points<>801 THEN
1800     DISP "Arrays are not dimensioned for this calibration"
1810     STOP
```

```
1820 END IF
1830 DISP "Uploading (querying) calibration arrays . . . ."
1840 !
1850 ! Turn off ASCII formatting on the I/O path.
1860 ASSIGN @Hp87xx;FORMAT OFF
1870 !
1880 ! Get the first error correction array.
1890 ENTER @Hp87xx;Corr1(*)
1900 !
1910 ! Turn on ASCII formatting.
1920 ASSIGN @Hp87xx;FORMAT ON
1930 !
1940 ! Get the "end of data" character.
1950 ENTER @Hp87xx;A$
1960 !
1970 ! For the reflection there are two more
1980 ! arrays to read.
1990 IF Arrays=3 THEN
2000 !
2010 ! Request and read in the second
2020 ! correction array.
2030     OUTPUT @Hp87xx;"TRAC? CH1SCORR2"
2040     Read_array(@Hp87xx,Corr2(*))
2050 !
2060 ! Request and read in the third
2070 ! correction array.
2080     OUTPUT @Hp87xx;"TRAC? CH1SCORR3"
2090     Read_array(@Hp87xx,Corr3(*))
2100 END IF
2110 DISP "Calibration arrays have been uploaded."
2120 WAIT 5
2130 DISP "Downloading (setting) calibration arrays . . . ."
2140 !
2150 ! Turn off correction before writing a
2160 ! calibration back into the analyzer.
2170 OUTPUT @Hp87xx;"SENS1:CORR:STAT OFF"
2180 !
2190 ! Set the number of points for the correction
2200 ! arrays.  (Not necessary in this example,
2210 ! but shown for emphasis.)
2220 OUTPUT @Hp87xx;"SENS1:SWE:POIN";Points
```

```
2230 !
2240 ! Prepare the analyzer to receive the first
2250 ! correction array in the indefinite block
2260 ! length format.
2270 OUTPUT @Hp87xx;"TRAC CH1SCORR1, #0";
2280 !
2290 ! Turn off ASCII formatting.
2300 ASSIGN @Hp87xx;FORMAT OFF
2310 !
2320 ! Send the first correction array to the
2330 ! analyzer.  The array transfer is
2340 ! terminated with the "END" signal.
2350 OUTPUT @Hp87xx;Corr1(*),END
2360 !
2370 ! Turn on ASCII formatting.
2380 ASSIGN @Hp87xx;FORMAT ON
2390 !
2400 ! For a reflection array download, there
2410 ! are two more arrays.
2420 IF Arrays=3 THEN
2430 !
2440 ! Prepare the analyzer to receive the
2450 ! 2nd array, then output it.
2460    OUTPUT @Hp87xx;"TRAC CH1SCORR2, ";
2470    Write_array(@Hp87xx,Corr2(*))
2480 !
2490 ! Prepare the analyzer to receive the
2500 ! 3rd array, then output it.
2510    OUTPUT @Hp87xx;"TRAC CH1SCORR3, ";
2520    Write_array(@Hp87xx,Corr3(*))
2530 END IF
2540 !
2550 ! Turn on the calibration just downloaded.
2560 OUTPUT @Hp87xx;"SENS1:CORR:STAT ON;*WAI"
2570 DISP "Calibration arrays have been downloaded."
2580 END
2590 !
2600 ! Subprogram for reading binary data array from
2610 ! the analyzer.  The command requesting a specific
2620 ! data array has already been sent prior to
2630 ! calling this subprogram.
```

```
2640 !
2650 SUB Read_array(@Hp87xx,INTEGER Array(*))
2660     DIM A$[10]
2670     INTEGER Digits,Bytes
2680     ASSIGN @Hp87xx;FORMAT ON
2690     ENTER @Hp87xx USING "%,A,D";A$,Digits
2700     ENTER @Hp87xx USING "%,"&VAL$(Digits)&"D";Bytes
2710     ASSIGN @Hp87xx;FORMAT OFF
2720     ENTER @Hp87xx;Array(*)
2730     ASSIGN @Hp87xx;FORMAT ON
2740     ENTER @Hp87xx;A$
2750 SUBEND
2760 !
2770 ! Subprogram for writing binary data array to
2780 ! the analyzer.  The command requesting a specific
2790 ! data array has already been sent prior to
2800 ! calling this subprogram.
2810 !
2820 SUB Write_array(@Hp87xx,INTEGER Array(*))
2830     OUTPUT @Hp87xx;"#0";
2840     ASSIGN @Hp87xx;FORMAT OFF
2850     OUTPUT @Hp87xx;Array(*),END
2860     ASSIGN @Hp87xx;FORMAT ON
2870 SUBEND
2880 !
2890 !****************************************************************
2900 ! Iden_port:   Identify io port to use.
2910 ! Description: This routines sets up the I/O port address for
2920 !              the SCPI interface.  For "HP 87xx" instruments,
2930 !              the address assigned to @Hp87xx = 800 otherwise,
2940 !              716.
2950 !****************************************************************
```

```
2960 SUB Iden_port
2970     COM /Sys_state/ @Hp87xx,Scode
2980 !
2990     IF POS(SYSTEM$("SYSTEM ID"),"HP 87")<>0 THEN
3000         ASSIGN @Hp87xx TO 800
3010         Scode=8
3020     ELSE
3030         ASSIGN @Hp87xx TO 716
3040         Scode=7
3050     END IF
3060 !
3070 SUBEND !Iden_port
3080 !
```

CALKIT Example Program

This instrument state file demonstrates the type of file required to download user-defined calibration kits. To see an example of using this feature, refer to "Writing or Editing Your Own Cal Kit File", in Chapter 6 of the *User's Guide.*

```
10  !$  Standard Definitions for HP 85054B Precision
        Type-N Cal Kit.
11  !
12  !$  This is a Cal Kit definition file, which
13  !$  uses the same format as a BASIC program.
14  !$  Lines that contain "!$" are comments.
15  !$
16  !$  Put your Cal Kit file on a disk, and use the
17  !$  analyzer's [SAVE/RECALL] [Recall State] keys
18  !$  to load your custom Cal Kit into the analyzer.
20  !
30  !$  Definitions for 50 Ohm jack (FEMALE center
        contact) test
40  !$  ports, plug (MALE center contact) standards.
50  !
60  ! OPEN:  $ HP 85054-60027 Open Circuit Plug
70  !    Z0  50.0 $ Ohms
80  !    DELAY  57.993E-12 $ Sec
90  !    LOSS  0.8E+9 $ Ohms/Sec
100 !    C0  88.308E-15 $ Farads
110 !    C1  1667.2E-27 $ Farads/Hz
120 !    C2 -146.61E-36 $ Farads/Hz^2
130 !    C3  9.7531E-45 $ Farads/Hz^3
140 !
150 ! SHORT:  $ HP 85054-60025 Short Circuit Plug
160 !    Z0  50.0 $ Ohms
170 !    DELAY  63.078E-12 $ Sec
180 !    LOSS  8.E+8 $ Ohms/Sec
190 !
200 ! LOAD:  $ HP 00909-60011 Broadband Load Plug
210 !    Z0  50.0 $ Ohms
220 !    DELAY  0.0 $ Sec
230 !    LOSS   0.0 $ Ohms/Sec
```

```
240 !
250 ! THRU:  $ HP 85054-60038 Plug to Plug Adapter
260 !    ZO  50.0 $ Ohms
270 !    DELAY  196.0E-12 $ Sec
280 !    LOSS   2.2E+9 $ Ohms/Sec
290 !
300 END
```

Instrument State and Save/Recall

LEARNSTR	Using the learn string to upload and download instrument states.
SAVERCL	Saving and recalling instrument states, calibrations and data. The example also demonstrates saving data in an ASCII file that includes both magnitude and frequency information.

LEARNSTR Example Program

This program demonstrates how to upload and download instrument states using the learn string. The learn string is a fast and easy way to read an instrument state. It is read out using the `*LRN?` query (an IEEE 488.2 common commands). To restore the learn string simply output the string to the analyzer.

The learn string contains a mnemonic at the beginning that tells the analyzer to restore the instrument state.

The learn string is transferred as a block. The header is ASCII formatted and the data is in the instrument's internal binary format. The number of bytes in the block of data is determined by the instrument state (no more than 20000 bytes).

```
"SYST:SET #<digits><bytes><learn string data>"
```

The "long" learnstring, in addition to the instrument state like the normal learnstring, will include data and calibration arrays IF they are selected using the Define Save function under SAVE/RECALL. The SCPI equivalent command for saving the calibration arrays is added before the "long" learnstring query.

```
1000 !Filename:  LEARNSTR
1010 !
1020 ! Description:
1030 !   1. Query the learn string.
1040 !   2. Preset the analyzer.
1050 !   3. Send the learn string,
1060 !      restoring the previous state.
1070 !
1080 DIM Learnstr$[20000]
1090 !
1100 COM /Sys_state/ @Hp87xx,Scode
1110 ! Identify I/O Port
1120 CALL Iden_port
1130 !
1140 !
1150 ! Request the learnstring.  If the "long"
1160 ! learnstring is desired, comment the line
1170 ! below, and uncomment the line after it.
1180 ! The "long" learnstring, in addition to
1190 ! the instrument state like the normal
1200 ! learnstring, will include data and
1210 ! calibration arrays IF they are selected
1220 ! using the Define Save function under
1230 ! SAVE RECALL.  The SCPI equivalent command
1240 ! for saving the calibration arrays is
1250 ! added before the "long" learnstring query.
1260 OUTPUT @Hp87xx;"*LRN?"
1270 ! OUTPUT @Hp87xx;"MMEM:STOR:STAT:CORR ON;:SYST:SET:LRNL?"
1280 !
1290 ! Read the learnstring from the analyzer.
1300 ! The USING "-K" format allows the data
1310 ! being transmitted to include characters
1320 ! (such as the line feed character) that
1330 ! would otherwise terminate the learnstring
1340 ! request prematurely.
1350 ENTER @Hp87xx USING "-K";Learnstr$
1360 DISP "Learn string has been read"
1370 WAIT 5
1380 !
1390 ! Preset the analyzer.
1400 OUTPUT @Hp87xx;"SYST:PRES;*OPC?"
```

```
1410 !
1420 ! Wait for the preset operation to complete.
1430 ENTER @Hp87xx;Opc
1440 DISP "Instrument has been PRESET"
1450 WAIT 5
1460 !
1470 ! Output the learnstring to the analyzer.
1480 ! The mnemonic is included in the string,
1490 ! so no command preceding "Learnstr$" is
1500 ! necessary.
1510 OUTPUT @Hp87xx;Learnstr$
1520 DISP "Instrument state has been restored"
1530 END
1540 !
1550 !***************************************************************
1560 ! Iden_port:   Identify io port to use.
1570 ! Description: This routines sets up the I/O port address for
1580 !              the SCPI interface.  For "HP 87xx" instruments,
1590 !              the address assigned to @Hp87xx = 800 otherwise,
1600 !              716.
1610 !***************************************************************
1620 SUB Iden_port
1630    COM /Sys_state/ @Hp87xx,Scode
1640 !
1650    IF POS(SYSTEM$("SYSTEM ID"),"HP 87")<>0 THEN
1660       ASSIGN @Hp87xx TO 800
1670       Scode=8
1680    ELSE
1690       ASSIGN @Hp87xx TO 716
1700       Scode=7
1710    END IF
1720 !
1730 SUBEND !Iden_port
1740 !
```

SAVERCL Example Program

This program demonstrates how to save instrument states, calibrations and data to a mass storage device. The device used in this example is the analyzer's internal 3.5" disk drive. The only change needed to use this program with the internal non-volatile memory is to change the mass storage unit specifier.

The three choices are the internal floppy disk drive (`INT:`), the internal non-volatile memory, (`MEM:`), and the internal volatile memory, (`RAM:`).

Lines `1110-1320` are an example of saving an instrument state and calibration on the internal floppy disk drive.

Lines `1460-1470` are an example of recalling that instrument state and calibration.

Lines `1510-1560` are an example of saving a data trace (magnitude and frequency values) to an ASCII formatted file on the internal floppy disk drive. This file cannot be recalled into the instrument. It can, however, be imported directly into spreadsheets and word processors.

```
1000 !Filename:  SAVERCL
1010 !
1020 !
1030 COM /Sys_state/ @Hp87xx,Scode
1040 ! Identify I/O Port
1050 CALL Iden_port
1060 !
1070 !
1080 ! Select the internal floppy disk drive
1090 ! as the mass storage device.
1100 OUTPUT @Hp87xx;"MMEM:MSIS 'INT:'"
1110 !
1120 ! Turn on the saving of the instrument state
1130 ! as part of the "Define Save" function under
1140 ! SAVE RECALL.
1150 OUTPUT @Hp87xx;"MMEM:STOR:STAT:IST ON"
1160 !
1170 ! Turn on the saving of the calibration
1180 ! as part of the "Define Save" function under
```

```
1190 ! SAVE RECALL.
1200 OUTPUT @Hp87xx;"MMEM:STOR:STAT:CORR ON"
1210 !
1220 ! Turn off the saving of the data
1230 ! as part of the "Define Save" function under
1240 ! SAVE RECALL.
1250 OUTPUT @Hp87xx;"MMEM:STOR:STAT:TRAC OFF"
1260 !
1270 ! Save the current defined state (STAT 1) into
1280 ! a file named "FILTER".  Use *OPC? to make
1290 ! sure the operation is completed before any
1300 ! other operation begins.
1310 OUTPUT @Hp87xx;"MMEM:STOR:STAT 1,'FILTER';*OPC?"
1320 ENTER @Hp87xx;Opc
1330 DISP "Instrument state and calibration have been saved"
1340 !
1350 ! Preset the instrument so that the change in state
1360 ! is easy to see when it is recalled.
1370 OUTPUT @Hp87xx;"SYST:PRES;*OPC?"
1380 ENTER @Hp87xx;Opc
1390 DISP "Instrument has been PRESET"
1400 WAIT 5
1410 !
1420 ! Recall the file "FILTER" from the internal
1430 ! floppy disk drive.  This becomes the new instrument
1440 ! state.  Use of the *OPC query allows hold off of
1450 ! further commands until the analyzer is reconfigured.
1460 OUTPUT @Hp87xx;"MMEM:LOAD:STAT 1,'INT:FILTER';*OPC?"
1470 ENTER @Hp87xx;Opc
1480 !
1490 ! Take a single sweep to ensure that valid measurement
1500 ! data is acquired.
1510 OUTPUT @Hp87xx;"ABOR;:INIT:CONT OFF;:INIT;*WAI"
1520 DISP "Instrument state and calibration have been recalled"
1530 !
1540 ! Save that measurement data into an ASCII file.
1550 ! called "DATA0001" on the internal floppy disk drive.
1560 OUTPUT @Hp87xx;"MMEM:STOR:TRAC CH1FDATA,'INT:DATA0001'"
1570 DISP "Data has been saved (ASCII format)"
1580 END
1590 !
```

```
1600 !**************************************************************
1610 ! Iden_port:   Identify io port to use.
1620 ! Description: This routines sets up the I/O port address for
1630 !              the SCPI interface.  For "HP 87xx" instruments,
1640 !              the address assigned to @Hp87xx = 800 otherwise,
1650 !              716.
1660 !**************************************************************
1670 SUB Iden_port
1680     COM /Sys_state/ @Hp87xx,Scode
1690 !
1700     IF POS(SYSTEM$("SYSTEM ID"),"HP 87")<>0 THEN
1710         ASSIGN @Hp87xx TO 800
1720         Scode=8
1730     ELSE
1740         ASSIGN @Hp87xx TO 716
1750         Scode=7
1760     END IF
1770 !
1780 SUBEND !Iden_port
1790 !
```

Hardcopy Control

PRINTPLT	Using the serial and parallel ports for hardcopy output. The example also demonstrates plotting test results to an HPGL file.
PASSCTRL	Using pass control and the HP-IB for hardcopy output. The example uses an HP-IB printer.
FAST_PRT	Provides fast graph dumps to PCL5 printers.

PRINTPLT Example Program

This program demonstrates how to send a hardcopy to a printer on the serial interface. This is done by selecting the appropriate device, setting up the baud rate and hardware handshaking, and sending the command to print or plot. The `*OPC?` query is used in this example to indicate when the printout is complete. Another method of obtaining the same results is to monitor the Hardcopy in Progress bit (bit 9 in the Operational Status Register). More information on printing or plotting is available in the *User's Guide*.

Lines `1170-1400` demonstrate sending a hardcopy output to a printer connected to the serial port. The same program could be used to send hardcopy output to a device on the parallel port. The only changes would be deleting lines `1230-1280` and changing line `1200` to read `HCOP:DEV:PORT PAR`.

Lines `1430-1680` demonstrate how to create an HPGL file (plotter language) and send it to the disk in the internal 3.5" disk drive.

```
1000 !Filename:  PRINTPLT
1010 !
1020 ! Description:
1030 !   1. Select serial port.  Configure it.
1040 !   2. Dump table of trace values.
1050 !   3. Re-configure hardcopy items to dump.
1060 !   4. Dump HP-GL file to internal floppy.
1070 !
1080 !
1090 COM /Sys_state/ @Hp87xx,Scode
1100 ! Identify I/O Port
1110 CALL Iden_port
1120 !
1130 !
1140 ! Select the output language (PCL-Printer
1150 ! Control Language) and the hardcopy port
1160 ! to serial.
1170 OUTPUT @Hp87xx;"HCOP:DEV:LANG PCL;PORT SER"
1180 !
1190 ! Select baud rate to 19200.
1200 OUTPUT @Hp87xx;"SYST:COMM:SER:TRAN:BAUD 19200"
```

```
1210 !
1220 ! Select the handshaking protocol to Xon/Xoff.
1230 OUTPUT @Hp87xx;"SYST:COMM:SER:TRAN:HAND XON"
1240 !
1250 ! Select the type of output to table, which
1260 ! is the same as the softkey List Trace
1270 ! Values under the Define Hardcopy menu.
1280 OUTPUT @Hp87xx;"HCOP:DEV:MODE TABL"
1290 !
1300 ! Send the command to start a hardcopy, and
1310 ! use *OPC query to make sure the hardcopy is
1320 ! complete before continuing.
1330 OUTPUT @Hp87xx;"HCOP;*OPC?"
1340 ENTER @Hp87xx;Opc
1350 DISP "Hardcopy to serial printer - COMPLETE!"
1360 !
1370 ! Select the HPGL language and the hardcopy
1380 ! port to be the currently selected mass memory
1390 ! device.
1400 OUTPUT @Hp87xx;"HCOP:DEV:LANG HPGL;PORT MMEM"
1410 !
1420 ! Include trace data in the plot.
1430 OUTPUT @Hp87xx;"HCOP:ITEM:TRAC:STAT ON"
1440 !
1450 ! Turn graticule off in the hardcopy dump.
1460 OUTPUT @Hp87xx;"HCOP:ITEM:GRAT:STAT OFF"
1470 !
1480 ! Include frequency and measurement
1490 ! annotation.
1500 OUTPUT @Hp87xx;"HCOP:ITEM:ANN:STAT ON"
1510 !
1520 ! Include marker symbols.
1530 OUTPUT @Hp87xx;"HCOP:ITEM:MARK:STAT ON"
1540 !
1550 ! Include title (and/or time/date if
1560 ! already selected).
1570 OUTPUT @Hp87xx;"HCOP:ITEM:TITL:STAT ON"
1580 !
1590 ! Define the hardcopy to be both the graph
1600 ! and a marker table.
1610 OUTPUT @Hp87xx;"HCOP:DEV:MODE GMAR"
```

```
1620 !
1630 ! Send the command to plot and use *OPC
1640 ! query to wait for finish.
1650 OUTPUT @Hp87xx;"HCOP;*OPC?"
1660 ENTER @Hp87xx;Opc
1670 DISP "Plot to floppy disk - COMPLETE!"
1680 END
1690 !
1700 !************************************************************
1710 ! Iden_port:   Identify io port to use.
1720 ! Description: This routines sets up the I/O port address for
1730 !              the SCPI interface.  For "HP 87xx" instruments,
1740 !              the address assigned to @Hp87xx = 800 otherwise,
1750 !              716.
1760 !************************************************************
1770 SUB Iden_port
1780     COM /Sys_state/ @Hp87xx,Scode
1790 !
1800     IF POS(SYSTEM$("SYSTEM ID"),"HP 87")<>0 THEN
1810         ASSIGN @Hp87xx TO 800
1820         Scode=8
1830     ELSE
1840         ASSIGN @Hp87xx TO 716
1850         Scode=7
1860     END IF
1870 !
1880 SUBEND !Iden_port
1890 !
```

PASSCTRL Example Program

This program demonstrates how to send a hardcopy to an HP-IB printer. This is done by passing active control of the bus to the analyzer so it can control the printer. More information about passing control to the analyzer is available in Chapter 3, "Passing Control."

```
1000 !Filename:  PASSCTRL
1010 !
1020 ! Description:
1030 !   External controller runs this program, which
1040 !   instructs the analyzer to perform a hardcopy
1050 !   and then passes control to the analyzer.
1060 !   Analyzer performs hardcopy over HP-IB
1070 !   to printer at 701, then passes control back.
1080 !
1090 !   This program only works on controllers which
1100 !   implement pass control properly.  HP s700
1110 !   computers running BASIC-UX 7.0x will need
1120 !   to upgrade to a newer BASIC-UX version.
1130 !
1140 !
1150 COM /Sys_state/ @Hp87xx,Scode,Internal
1160 ! Identify I/O Port
1170 CALL Iden_port
1180 !
1190 !
1200 ! Select the language to PCL (Printer
1210 ! Control Language) and the output port
1220 ! to HP-IB.
1230 OUTPUT @Hp87xx;"HCOP:DEV:LANG PCL;PORT GPIB"
1240 !
1250 ! Select the HP-IB address for the hardcopy
1260 ! device on the HP-IB.
1270 OUTPUT @Hp87xx;"SYST:COMM:GPIB:HCOP:ADDR 1"
1280 !
1290 ! Set the output to graph only.
1300 OUTPUT @Hp87xx;"HCOP:DEV:MODE GRAP"
1310 !
```

```
1320 ! If the internal controller is being used...
1330 IF Internal=1 THEN
1340 !
1350 ! then make it System Controller of HP-IB
1360     OUTPUT @Hp87xx;"SYST:COMM:GPIB:CONT ON"
1370 END IF
1380 !
1390 ! Clear Status Registers
1400 OUTPUT @Hp87xx;"*CLS"
1410 !
1420 ! Enable the Request Control bit in the Event
1430 ! Status Register.
1440 OUTPUT @Hp87xx;"*ESE 2"
1450 !
1460 ! Clear the Service Request enable register;
1470 ! SRQ is not being used.
1480 OUTPUT @Hp87xx;"*SRE 0"
1490 !
1500 ! Send the hardcopy command to start the
1510 ! print.
1520 OUTPUT @Hp87xx;"HCOP"
1530 LOOP
1540 !
1550 ! Read the status byte using Serial Poll.
1560     Stat=SPOLL(@Hp87xx)
1570 !
1580 ! Exit when the analyzer requests active control
1590 ! of HP-IB from the system controller.
1600 EXIT IF BIT(Stat,5)=1
1610 END LOOP
1620 !
1630 ! Now system controller passes control to
1640 ! the analyzer.
1650 PASS CONTROL @Hp87xx
1660 DISP "Hardcopy in Progress...";
1670 IF Internal=1 THEN
1680 ! If using the internal IBASIC controller,
1690 ! then use the *OPC query method to wait
1700 ! for hardcopy completion.
1710     OUTPUT @Hp87xx;"*OPC?"
1720     ENTER @Hp87xx;Opc
```

```
1730 ELSE
1740 ! If external computer control, then...
1750    LOOP
1760 !
1770 ! Monitor the HP-IB status in the
1780 ! external computer's HP-IB status
1790 ! register.  Here, the HP-IB interface
1800 ! code 7 register 6 status is requested
1810 ! and put into "Hpib".
1820        DISP ".";
1830        WAIT 1! No need to poll rapidly
1840        STATUS 7,6;Hpib
1850 !
1860 ! When active control is returned to the
1870 ! system controller (bit 6 set), then exit.
1880 ! (This fails on s700s running BASIC 7.0x)
1890    EXIT IF BIT(Hpib,6)=1
1900    END LOOP
1910 END IF
1920 DISP "HARDCOPY COMPLETE!"
1930 END
1940 !
1950 !***************************************************************
1960 ! Iden_port:   Identify io port to use.
1970 ! Description: This routines sets up the I/O port address for
1980 !              the SCPI interface.  For "HP 87xx" instruments,
1990 !              the address assigned to @Hp87xx = 800 otherwise,
2000 !              716.
2010 !***************************************************************
```

```
2020 SUB Iden_port
2030     COM /Sys_state/ @Hp87xx,Scode,Internal
2040 !
2050     IF POS(SYSTEM$("SYSTEM ID"),"HP 87")<>0 THEN
2060         ASSIGN @Hp87xx TO 800
2070         Scode=8
2080         Internal=1
2090     ELSE
2100         ASSIGN @Hp87xx TO 716
2110         Scode=7
2120         Internal=0
2130     END IF
2140 !
2150 SUBEND !Iden_port
2160 !
```

FAST_PRT Example Program

This program configures a PCL5 printer to accept HP-GL graphics commands from the analyzer. The program executes a hardcopy which causes the analyzer to send HP-GL commands to the parallel port PCL5 printer. Provides up to 10× speed improvement of some hardcopies.

```
1000 ! FAST_PRT
1010 !
1020 ! This program is designed to set up a PCL5 printer
1030 ! connected to the parallel port of the analyzer to
1040 ! accept HP-GL syntax.  HP-GL gives fast graph dumps.
1050 !
1060 ! Connect your PCL5 printer to the parallel printer of the
1070 ! analyzer, then run the program.
1075 !
1076 ! Note: Firmware hardcopy support for PCL5 for 871xCs can
1077 ! can be enabled by selecting a PCL5 harcopy device.
1078 ! This program may still be needed for the 871xBs.
1080 !
1090 ! Once the parallel printer has been configured to accept
1100 ! HPGL commands, a hardcopy is done, the printer is
1110 ! reset to normal mode, and the page is ejected.
1120 !
1130 DIM A$[50]
1140 !
1150 !
1160 COM /Sys_state/ @Hp87xx,Scode
1170 ! Identify I/O Port
1180 CALL Iden_port
1190 !
1200 ! Define the hardcopy device
1210 OUTPUT @Hp87xx;"HCOP:DEV:LANG HPGL;PORT CENT"
1220 !
1230 ! Define PCL5 escape codes needed to set up HPGL commands:
1240 DATA @E                    ! Reset, Eject page
1250 DATA &@12A                 ! Page size 8.5 x 11
1260 DATA &@aOL&@a4000M&@lOE ! No margins
1270 DATA @*c7400x5650y       ! 10.28 x 7.85 size 720/in
```

```
1280 !DATA @*c5500x5650y       ! if Marker table included
1290 !DATA @*c4255x3283y       ! portrait,remove Landscape Mode
1300 DATA &@l1O               ! Landscape Mode
1310 DATA @*p50x50y           ! Cursor to anchor point
1320 DATA @*c0T               ! Set picture anchor point
1330 DATA @*r-3U              ! CMY Palette
1340 !DATA @*r1U               ! Monochrome optional
1350 DATA @%1B                ! HPGL Mode
1360 DATA $                   ! dump plot
1370 DATA @%0A                ! Exit HPGL Mode
1380 DATA @E                  ! Eject page
1390 DATA DONE                ! End of defined escape codes
1400 !
1410 ! Send the defined escape codes to the printer
1420 LOOP
1430     READ A$
1440 EXIT IF A$="DONE"
1450     FOR I=1 TO LEN(A$)
1460         SELECT A$[I;1]
1470         CASE "@"! Escape Character
1480             OUTPUT @Hp87xx;"DIAG:PORT:WRITE 15,0,27"
1490         CASE "$"! Dump the plot
1500             OUTPUT @Hp87xx;"HCOP;*WAI"
1510         CASE ELSE! Send Character
1520             OUTPUT @Hp87xx;"DIAG:PORT:WRITE 15,0,";NUM(A$[I;1])
1530         END SELECT
1540     NEXT I
1550 END LOOP
1560 !
1570 END
1580 !
1590 !**************************************************************
1600 ! Iden_port:   Identify io port to use.
1610 ! Description: This routines sets up the I/O port address for
1620 !              the SCPI interface.  For "HP 87xx" instruments,
1630 !              the address assigned to @Hp87xx = 800 otherwise,
1640 !              716.
1650 !**************************************************************
```

```
1660 SUB Iden_port
1670     COM /Sys_state/ @Hp87xx,Scode
1680 !
1690     IF POS(SYSTEM$("SYSTEM ID"),"HP 87")<>0 THEN
1700         ASSIGN @Hp87xx TO 800
1710         Scode=8
1720     ELSE
1730         ASSIGN @Hp87xx TO 716
1740         Scode=7
1750     END IF
1760 !
1770 SUBEND !Iden_port
1780 !
```

Service Request

SRQ	Generating a service request interrupt. The example uses the status reporting structure to generate an interrupt as soon as averaging is complete.
SRQ_INT	Monitoring the status report of the analyzer.

SRQ Example Program

This program demonstrates generating a service request interrupt. The SRQ is used to indicate when averaging is complete. More information on service requests and the status registers is available in Chapter 5, "Using Status Registers."

In this program, the `STATus:PRESet` executed in line 1250 has the effect of setting all bits in the averaging status transition registers (positive transitions to 0, negative transitions to 1). It also sets up the operational status transition registers (positive transitions to 1, negative transitions to 0). These are the states needed to generate an interrupt when averaging is complete.

```
1000 !Filename:  SRQ
1010 !
1020 ! Description:
1030 !   Set an SRQ to occur when averaging is complete.
1040 !   Turn on averaging, and set to 8 averages.
1050 !   Initiate sweeps.  SRQ will occur after 8 sweeps.
1060 !   Wait in a do-nothing loop, checking SRQ flag.
1070 !   Display message after SRQ flag is set.
1080 !
1090 !
1100 COM /Sys_state/ @Hp87xx,Scode
1110 ! Identify I/O Port
1120 CALL Iden_port
1130 !
1140 !
1150 ! Clear status registers.
1160 OUTPUT @Hp87xx;"*CLS"
1170 !
1180 ! Clear the Service Request Enable register.
1190 OUTPUT @Hp87xx;"*SRE 0"
1200 !
1210 ! Clear the Standard Event Status Enable register.
1220 OUTPUT @Hp87xx;"*ESE 0"
1230 !
1240 ! Preset the remaining status registers.
1250 OUTPUT @Hp87xx;"STAT:PRES"
```

```
1260 !
1270 ! Set operation status register to report
1280 ! to the status byte on POSITIVE transition of
1290 ! the averaging bit.
1300 OUTPUT @Hp87xx;"STAT:OPER:ENAB 256"
1310 !
1320 ! Set averaging status register to report to
1330 ! operational status register on NEGATIVE transition
1340 ! of the averaging done bits.  The NEGATIVE
1350 ! transition needs to be detected because the
1360 ! averaging bit 0 is set to 1 while the analyzer
1370 ! is sweeping on channel 1 and the number of
1380 ! sweeps completed since averaging restart is
1390 ! less than the averaging factor.  When the bit
1400 ! goes back to 0, the averaging is done.
1410 OUTPUT @Hp87xx;"STAT:OPER:AVER:ENAB 1"
1420 !
1430 ! Enable the operational status bit in the status
1440 ! byte to generate an SRQ.
1450 OUTPUT @Hp87xx;"*SRE 128"
1460 !
1470 ! On an interrupt from HP-IB "Scode" (Interface
1480 ! Select Code) SRQ bit (2), branch to the interrupt
1490 ! service routine "Srq_handler".
1500 ON INTR Scode,2 GOSUB Srq_handler
1510 !
1520 ! Now enable the interrupt on SRQ (Service Request).
1530 ENABLE INTR Scode;2
1540 !
1550 ! Set averaging factor to 8.
1560 OUTPUT @Hp87xx;"SENS1:AVER:COUN 8;*WAI"
1570 !
1580 ! Turn on averaging and restart.
1590 OUTPUT @Hp87xx;"SENS1:AVER ON;AVER:CLE;*WAI"
1600 !
1610 ! Turn on continuous sweep trigger mode.
1620 OUTPUT @Hp87xx;"ABOR;:INIT1:CONT ON;*WAI"
1630 !
1640 ! Initialize flag indicating when averaging done
1650 ! to 0.  Then loop continuously until the
1660 ! interrupt is detected, and the interrupt
```

```
1670 ! service routine acknowledges the
1680 ! interrupt and sets the flag to 1.
1690 Avg_done=0
1700 DISP "Waiting for SRQ on averaging complete.";
1710 LOOP
1720     DISP ".";
1730     WAIT .1! Slow down dots
1740 EXIT IF Avg_done=1
1750 END LOOP
1760 !
1770 ! Display desired completion message.
1780 DISP
1790 DISP "Got SRQ.  Averaging Complete!"
1800 STOP
1810 !
1820 Srq_handler: ! Interrupt Service Routine
1830 !
1840 ! Determine that the analyzer was actually
1850 ! the instrument that generated the
1860 ! interrupt.
1870 Stb=SPOLL(@Hp87xx)
1880 !
1890 ! Determine if the operation status register
1900 ! caused the interrupt by looking at bit 7
1910 ! of the result of the serial poll.
1920 IF BINAND(Stb,128)<>0 THEN
1930 !
1940 ! Read the operational status event register.
1950     OUTPUT @Hp87xx;"STAT:OPER:EVEN?"
1960     ENTER @Hp87xx;Op_event
1970 !
1980 ! Determine if the averaging status register
1990 ! bit 8 is set.
2000     IF BINAND(Op_event,256)<>0 THEN
2010 !
2020 ! If so, then set flag indicating
2030 ! averaging done.
2040         Avg_done=1
2050     END IF
2060 END IF
2070 RETURN
```

```
2080 END
2090 !
2100 !************************************************************
2110 ! Iden_port:   Identify io port to use.
2120 ! Description: This routines sets up the I/O port address for
2130 !              the SCPI interface.  For "HP 87xx" instruments,
2140 !              the address assigned to @Hp87xx = 800 otherwise,
2150 !              716.
2160 !************************************************************
2170 SUB Iden_port
2180     COM /Sys_state/ @Hp87xx,Scode
2190 !
2200     IF POS(SYSTEM$("SYSTEM ID"),"HP 87")<>0 THEN
2210         ASSIGN @Hp87xx TO 800
2220         Scode=8
2230     ELSE
2240         ASSIGN @Hp87xx TO 716
2250         Scode=7
2260     END IF
2270 !
2280 SUBEND !Iden_port
2290 !
```

SRQ_INT Example Program

This program demonstrates how to monitor the status report of the network analyzer via an interrupt handler. It monitors all the status or error bits of the status register. Whenever an error or an event occurs, the analyzer will interrupt this program. This program will then decode the error bits and display the appropriate messages. For a detailed status report register map, refer to Chapter 5, "Using Status Registers."

```
1000 !  SRQ_INT
1010 !  BASIC program:
1020 !
1030 !****************************************************************
1040 !  Description:  This program implements the interrupt handler
1050 !  routine for SRQ status reports.  It tries to decode all the
1060 !  possible error bits by rippling it through the registers
1070 !  and print out the appropriate messages for the status report.
1080 !  Refer to the status model diagram for registers mapping.
1090 !
1100 !  Note:  To setup additional states other than the status
1110 !         model, add code to the subroutine "Setup_states".
1120 !****************************************************************
1130 !
1140 !************ Main program **************
1150 !
1160 !  Make @Hp87xx common to all subroutines
1170 COM /Sys_state/ @Hp87xx,Scode
1180 ! Identify the computer we are running on
1190 ! and assign the i/o port address to @Hp87xx
1200 CALL Iden_port
1210 !
1220 ! Setup all required SRQ registers
1230 CALL Setup_srq_regs
1240 !
1250 ! This is required if user wants to detect either
1260 ! "Any Ext. Keybd. Pressed" or "Front Panel Knob Turned"
1270 ! of the Device Status Register.
1280 OUTPUT @Hp87xx;"SYST:KEY:QUE:STATE ON"
1290 !
```

```
1300 ! Go to subroutine to setup any neccessary states
1310 CALL Setup_states
1320 !
1330 Report_count=0
1340 ! Forever loop to wait for any failed events to happened
1350 DISP "Waiting for any Failed Events.........Report Count =
     ";Report_count
1360 Do_loop: !
1370 GOSUB Set_userbit                          ! Read and display
     variables
1380 !
1390 GOTO Do_loop
1400 STOP
1410 !
1420 !************ Subroutine Blocks ********************
1430 !
1440 !*************************************************************
1450 ! Set_userbit:  Setup user bit.
1460 ! Description:  This subroutine waits for an SRQ interrupt to
1470 ! signal that a sweep has finished.  It then clears the HP-IB
1480 ! registers by reading them.  Once that is done, the user bit
1490 ! is toggled.
1500 !*************************************************************
1510 !
1520 Set_userbit:!
1530 !
1540 ON INTR Scode GOTO Read_results       ! Set up interrupt branching
1550 ENABLE INTR Scode;2                   ! Allow interrupt on SRQ
1560 Suspend: !WAIT 5                      ! Use WAIT 'n' to suspend IBASIC
1570 GOTO Suspend
1580 !
1590 ! Interrupt Service Routine
1600 Read_results:                         ! Program has finished
1610 A=SPOLL(@Hp87xx)                      !  and clear the SRQ
1620 !
1630 ! This CLEAR command is for clearing out the bus just in case
1640 ! this is a Query error.  Without this CLEAR command, the previous
1650 ! Query would screw up the state of the instrument and the next
1660 ! Query will get an error.
1670 CLEAR @Hp87xx
1680 OUTPUT @Hp87xx;"*STB?"
```

```
1690 ENTER @Hp87xx;Stbr
1700 WHILE Stbr<>0
1710     CALL Decode_srq(Stbr)
1720     Stb=Stbr
1730     OUTPUT @Hp87xx;"*STB?"
1740     ENTER @Hp87xx;Stbr
1750 END WHILE
1760 OUTPUT @Hp87xx;"*CLS"   ! Clear status byte
1770 BEEP
1780 Report_count=Report_count+1
1790 DISP "Waiting for any Failed Events.........Report Count =
     ";Report_count
1800 !
1810 RETURN
1820 END
1830 !
1840 !***********   Status Report Decode Block ********************
1850 !
1860 !
1870 !*************************************************************
1880 ! Decode_srq:  Decode status byte.
1890 ! Description: Decode the Srq register and ripple through the
1900 !              registers and decode the necessary failed registers.
1910 !              The decoding is done with the Event registers.  The
1920 !              corresponding Condition Registers are only read and
1930 !              display.  The numbers are display in hex numbers
1940 !              with a leading "0x".
1950 !              If any Event has failed, a message will be display
1960 !              with the "Meas" Channel number and "Segment" number.
1970 !*************************************************************
1980 SUB Decode_srq(Reg)
1990     COM /Sys_state/ @Hp87xx,Scode
2000     DIM Reg_name$(1:8)[40]
2010 !
2020 !
2030 !   Print out the Date of Time of this report
2040 !
2050     PRINT
2060     PRINT
2070     PRINT "Status Report:  ";DATE$(TIMEDATE);"  ";TIME$(TIMEDATE)
2080     PRINT
```

```
2090     PRINT "Status Byte = 0x";IVAL$(Reg,16)
2100     IF (BIT(Reg,6))=1 THEN
2110         RESTORE
2120         READ Reg_name$(*)
2130 !
2140 !       For each of the bit set in the status register,
2150 !       call Decode_reg() to decode the appropriate
2160 !       second level registers.
2170 !
2180         FOR I=2 TO 7
2190 !
2200             IF (BIT(Reg,I)=1) THEN
2210                 CALL Decode_reg(Reg_name$(I+1))
2220             END IF
2230 !
2240         NEXT I
2250 !
2260     ELSE
2270         PRINT "Bogus Interrupt??? Bit 6 of Status byte is not set???"
2280     END IF
2290 !
2300     PRINT
2310     PRINT ".......END OF REPORT"
2320     PRINT
2330 !
2340 !
2350 !   Status Register
2360     DATA  "Unknown Register"        ! Bit 0
2370     DATA  "Unknown Register"        ! Bit 1
2380     DATA  "Device Status"           ! Bit 2
2390     DATA  "Questionable"            ! Bit 3
2400     DATA  "Output Queue"            ! Bit 4
2410     DATA  "Standard Event"          ! Bit 5
2420     DATA  "Status Fail"             ! Bit 6
2430     DATA  "Operational"             ! Bit 7
2440 !
2450 SUBEND !DECODE_SRQ
2460 !
2470 !***************************************************************
2480 ! Decode_reg:  Decode the second level registers.
2490 ! Description: The Cases in the SELECT statements corresponds
```

```
2500 !              to the bits supported by the instrument.  Refer
2510 !              to the menu for the bit positions of these bits.
2520 !              For each failed event, the Event and Condition
2530 !              registers are read and Display.  The Event register
2540 !              is further used to decode the third level registers.
2550 !
2560 !*******************************************************************
2570 !
2580 SUB Decode_reg(Reg_name$)
2590     COM /Sys_state/ @Hp87xx,Scode
2600 !
2610     SELECT Reg_name$
2620 !
2630 !   Device Status register
2640     CASE "Device Status"! Bit 2
2650         OUTPUT @Hp87xx;"STAT:DEV:EVEN?"
2660         ENTER @Hp87xx;Dev_event
2670         PRINT "   Device Status Event Reg = 0x";IVAL$(Dev_event,16)
2680         OUTPUT @Hp87xx;"STAT:DEV:COND?"
2690         ENTER @Hp87xx;Dev_cond
2700         PRINT "   Device Status Condition Reg = 0x";IVAL$(Dev_cond,16)
2710         CALL Decode_dev(Dev_event)
2720 !
2730 !   Questionable status register
2740     CASE "Questionable"! Bit 3
2750 !
2760         OUTPUT @Hp87xx;"STAT:QUES:EVEN?"! Read and clear Questional
             STATUS reg.
2770         ENTER @Hp87xx;Ques_event
2780         PRINT "   Questionable Event Reg = 0x";IVAL$(Ques_event,16)
2790         OUTPUT @Hp87xx;"STAT:QUES:COND?"
2800         ENTER @Hp87xx;Ques_cond
2810         PRINT "   Questionable Condition Reg = 0x";IVAL$(Ques_cond,16)
2820         CALL Decode_ques(Ques_event)
2830 !
2840 !
2850     CASE "Standard Event"! Bit 4
2860 !
2870         OUTPUT @Hp87xx;"*ESR?"
2880         ENTER @Hp87xx;Stand_event
2890         PRINT "   Standard Event Reg = 0x";IVAL$(Stand_event,16)
```

```
2900         CALL Decode_esr(Stand_event)
2910 !
2920     CASE "Output Queue"! Bit 5
2930 !
2940         PRINT "   Message Available"
2950 !
2960 !   Latch bit of Status Byte register
2970     CASE "Status Fail"! Bit 6
2980 !     Do Nothing
2990 !
3000 !   Operational Status register
3010     CASE "Operational"! Bit 7
3020 !
3030         OUTPUT @Hp87xx;"STAT:OPER:EVEN?"!Read and clear Operational
             STATUS reg.
3040         ENTER @Hp87xx;Oper_event
3050         PRINT "   Operational Event Reg = 0x";IVAL$(Oper_event,16)
3060         OUTPUT @Hp87xx;"STAT:OPER:COND?"
3070         ENTER @Hp87xx;Oper_cond
3080         PRINT "   Operational Condition Reg = 0x";IVAL$(Oper_cond,16)
3090 !
3100         CALL Decode_oper(Oper_event)
3110 !
3120 !
3130     CASE ELSE
3140         PRINT "   Unsupported Bit set in Status Byte or ";
3150         PRINT "   Bogus interrupt. "
3160 !
3170     END SELECT
3180 SUBEND !DECODE_REG
3190 !
3200 !
3210 !************************************************************
3220 ! Decode_Ques:  Decode Questionable Fail register.
3230 ! Description:  Decode Questionable Fail register and Print out
3240 !               appropriate messages.
3250 !************************************************************
3260 SUB Decode_ques(Reg)
3270     COM /Sys_state/ @Hp87xx,Scode
3280     DIM Message$(0:15,0:1)[120]
3290     DIM Segment_event(4:7)
```

```
3300 !
3310     READ Message$(*)
3320     FOR I=0 TO 15
3330         IF Message$(I,0)="Enable" THEN
3340             IF BIT(Reg,I)=1 THEN
3350 !
3360                 PRINT Message$(I,1)
3370                 IF (I=9) THEN !  Check Limit Fail Register
3380                     OUTPUT @Hp87xx;"STAT:QUES:LIM:EVEN?"
3390                     ENTER @Hp87xx;Lim_event
3400                     PRINT "      Limit Fail Event Reg =
                         0x";IVAL$(Lim_event,16)
3410                     OUTPUT @Hp87xx;"STAT:QUES:LIM:COND?"
3420                     ENTER @Hp87xx;Lim_cond
3430                     PRINT "      Limit Fail Condition Reg =
                         0x";IVAL$(Lim_cond,16)
3440                     CALL Decode_lim(Lim_event)
3450                 END IF
3460 !
3470                 IF (I=4) OR (I=5) OR (I=6) OR (I=7) THEN
3480                     OUTPUT @Hp87xx;"STAT:QUES:SEGM";VAL$(I-3);":EVEN?"
3490                     ENTER @Hp87xx;Segment_event(I)
3500                     PRINT "      Segment ";VAL$(I-3);" Event Reg =
                         0x";IVAL$(Segment_event(I),16)
3510                     OUTPUT @Hp87xx;"STAT:QUES:SEGM";VAL$(I-3);":COND?"
3520                     ENTER @Hp87xx;Segment_cond
3530                     PRINT "      Segment ";VAL$(I-3);" Condition Reg =
                         0x";IVAL$(Segment_cond,16)
3540                     CALL Decode_seg(Segment_event(I),I-3)
3550                 END IF
3560             END IF
3570         END IF
3580     NEXT I
3590 !
3600 !  This array has two fields:
3610 !  First Field - If Enable, Display the next string message
3620 !                Else, ignore the next string message.
3630 !  Second Field - String message for the corresponding bits of
3640 !                 the Questionable register.
3650     DATA "Enable","      ALC UNLEVELED.............."      ! Bit 0
3660     DATA "Enable","      FREQ_ERROR................."      ! Bit 1
```

```
3670     DATA "Disable","      Bit 2 Unsupported"                ! Bit 2
3680     DATA "Disable","      Bit 3 Unsupported"                ! Bit 3
3690     DATA "Enable","      Segment 1 Limit Fail................."
         ! Bit 4
3700     DATA "Enable","      Segment 2 Limit Fail................."
         ! Bit 5
3710     DATA "Enable","      Segment 3 Limit Fail................."
         ! Bit 6
3720     DATA "Enable","      Segment 4 Limit Fail................."
         ! Bit 7
3730     DATA "Disable","      Bit 8 Unsupported"                ! Bit 8
3740     DATA "Enable","      Limit Fail................."       ! Bit 9
3750     DATA "Enable","      Stale Data(Data?).........."       ! Bit 10
3760     DATA "Disable","      Bit 11 Unsupported"               ! Bit 11
3770     DATA "Disable","      Bit 12 Unsupported"               ! Bit 12
3780     DATA "Disable","      Bit 13 Unsupported"               ! Bit 13
3790     DATA "Disable","      Bit 15 Unsupported"               ! Bit 14
3800     DATA "Disable","      Bit 15 Unsupported"               ! Bit 15
3810 SUBEND !Decode_Ques
3820 !
3830 !
3840 !
3850 !***************************************************************
3860 ! Decode_lim:  Decode Limit Fail register.
3870 ! Description: Decode Limit Fail register and Print out the
3880 !              appropriate messages.
3890 !***************************************************************
3900 !
3910 SUB Decode_lim(Reg)
3920     COM /Sys_state/ @Hp87xx,Scode
3930     DIM Message$(0:3)[120]
3940 !
3950     READ Message$(*)
3960 !
3970     FOR I=0 TO 3
3980         IF BIT(Reg,I)=1 THEN
3990             PRINT Message$(I)
4000         END IF
4010     NEXT I
4020 !
4030 !   Displaying message
```

```
4040     DATA "        Limit Line Failed   on  Meas 1"        ! Bit 0
4050     DATA "        Limit Line Failed   on  Meas 2"        ! Bit 1
4060     DATA "        Marker Limit Failed on  Meas 1"        ! Bit 2
4070     DATA "        Marker Limit Failed on  Meas 2"        ! Bit 3
4080 SUBEND !Decode_lim
4090 !
4100 !
4110 !**************************************************************
4120 ! Decode_seg:  Decode Segment status registers.
4130 ! Description: Decode Segment status registers and Print out the
4140 !              appropriate messages.
4150 !**************************************************************
4160 !
4170 SUB Decode_seg(Reg,Segment)
4180     COM /Sys_state/ @Hp87xx,Scode
4190     DIM Message$(0:9)[120]
4200 !
4210     READ Message$(*)
4220 !
4230 !   Check to see if bias regs need to be decoded.
4240     FOR I=0 TO 9
4250         IF BIT(Reg,I)=1 THEN
4260             PRINT Message$(I);Segment
4270             IF I=4 THEN ! Meas 1 has failed?
4280                 CALL Decode_bias("Meas 1",Segment)
4290             END IF
4300             IF I=5 THEN ! Meas 2 has failed?
4310                 CALL Decode_bias("Meas 2",Segment)
4320             END IF
4330         END IF
4340     NEXT I
4350 !
4360 !   Displaying message
4370     DATA "        Limit Line Failed     on Meas 1, Segment"       !
         Bit 0
4380     DATA "        Limit Line Failed     on Meas 2, Segment"       !
         Bit 1
4390     DATA "        Marker Limit Failed   on Meas 1, Segment"       !
         Bit 2
4400     DATA "        Marker Limit Failed   on Meas 2, Segment"       !
         Bit 3
```

```
4410    DATA "          Bias Limit Failed      on Meas 1, Segment"        !
        Bit 4
4420    DATA "          Bias Limit Failed      on Meas 2, Segment"        !
        Bit 5
4430    DATA "          Gain Bar Limit Failed  on Meas 1, Segment"        !
        Bit 6
4440    DATA "          Gain Bar Limit Failed  on Meas 2, Segment"        !
        Bit 7
4450    DATA "          IIc Ack Test Failed    on Meas 1, Segment"        !
        Bit 8
4460    DATA "          IIc Ack Test Failed    on Meas 2, Segment"        !
        Bit 9
4470 SUBEND !Decode_seg
4480 !
4490 !
4500 !***************************************************************
4510 ! Decode_dev:  Decode Device status register.
4520 ! Description: Decode Device status register and Print out the
4530 !              appropriate messages.
4540 !***************************************************************
4550 !
4560 SUB Decode_dev(Reg)
4570    COM /Sys_state/ @Hp87xx,Scode
4580    DIM Message$(0:3)[120]
4590 !
4600    READ Message$(*)
4610    FOR I=0 TO 3
4620        IF BIT(Reg,I)=1 THEN
4630            PRINT Message$(I)
4640        END IF
4650    NEXT I
4660 !
4670    DATA "      Any Key Pressed"
4680    DATA "      Any Softkey Pressed"
4690    DATA "      Any Ext. Keybd. Pressed"
4700    DATA "      Front Panel Knob Turned"
4710 SUBEND !Decode_dev_
4720 !
4730 !
4740 !
4750 !
```

```
4760 !************************************************************
4770 ! Decode_esr:  Decode Standard Event register
4780 ! Description: Decode Standard Event register and Print out the
4790 !              appropriate messages
4800 !************************************************************
4810 !
4820 SUB Decode_esr(Reg)
4830     COM /Sys_state/ @Hp87xx,Scode
4840     DIM Message$(0:7)[120]
4850 !
4860     READ Message$(*)
4870     FOR I=0 TO 7
4880         IF BIT(Reg,I)=1 THEN
4890             PRINT Message$(I)
4900             IF I=4 THEN
4910                 CALL Disp_err
4920             END IF
4930         END IF
4940     NEXT I
4950 !
4960     DATA "      Operation Complete"              ! Bit 0
4970     DATA "      Request Control"                 ! Bit 1
4980     DATA "      Qeury Error"                     ! Bit 2
4990     DATA "      Device-Dependent Error"          ! Bit 3
5000     DATA "      Execution Error"                 ! Bit 4
5010     DATA "      Command Error"                   ! Bit 5
5020     DATA "      User Request"                    ! Bit 6
5030     DATA "      Power On"                        ! Bit 7
5040 SUBEND !Decode_esr
5050 !
5060 !
5070 !************************************************************
5080 ! Decode_oper:  Decode Operational register.
5090 ! Description:  Decode Operational register and Print out the
5100 !               appropriate messages
5110 !************************************************************
5120 !
5130 SUB Decode_oper(Reg)
5140     COM /Sys_state/ @Hp87xx,Scode
5150     DIM Message$(0:15,0:1)[120]
5160 !
```

```
5170     READ Message$(*)
5180     FOR I=0 TO 15
5190         IF Message$(I,0)="Enable" THEN
5200             IF BIT(Reg,I)=1 THEN
5210 !
5220                 PRINT Message$(I,1)
5230                 IF I=4 THEN
5240 !
5250                     OUTPUT @Hp87xx;"STAT:OPER:MEAS:EVEN?"
5260                     ENTER @Hp87xx;Meas_event
5270                     PRINT "Measuring Event Reg                  =
                         Ox";IVAL$(Meas_event,16)
5280                     OUTPUT @Hp87xx;"STAT:OPER:MEAS:COND?"
5290                     ENTER @Hp87xx;Meas_cond
5300                     PRINT "Measuring Condition Reg              =
                         Ox";IVAL$(Meas_cond,16)
5310 !
5320                     CALL Decode_meas(Meas_event)
5330                 ELSE
5340                     IF I=8 THEN
5350 !
5360                         OUTPUT @Hp87xx;"STAT:OPER:AVER:EVEN?"
5370                         ENTER @Hp87xx;Aver_event
5380                         PRINT "Averaging Event Reg                  =
                             Ox";IVAL$(Aver_event,16)
5390                         OUTPUT @Hp87xx;"STAT:OPER:AVER:COND?"
5400                         ENTER @Hp87xx;Aver_cond
5410                         PRINT "Averaging Condition Reg              =
                             Ox";IVAL$(Aver_cond,16)
5420 !
5430                         CALL Decode_avg(Aver_event)
5440                     END IF
5450                 END IF
5460 !
5470             END IF
5480         END IF
5490     NEXT I
5500 !
5510 !   This array has two fields:
5520 !   First Field - If Enable, Display the next string message
5530 !                 Else, ignore the next string message.
```

```
5540 !  Second Field - String message for the corresponding bits of
5550 !                 the Questionable register.
5560    DATA "Enable","      Calibrating.............."       ! Bit 0
5570    DATA "Enable","      Settling................."       ! Bit 1
5580    DATA "Disable","      Bit 2 Unsupported"              ! Bit 2
5590    DATA "Disable","      Bit 3 Unsupported"              ! Bit 3
5600    DATA "Enable","      Measuring................"       ! Bit 4
5610    DATA "Disable","      Bit 5 Unsupported"              ! Bit 5
5620    DATA "Disable","      Bit 6 Unsupported"              ! Bit 6
5630    DATA "Enable","      Correcting..............."       ! Bit 7
5640    DATA "Enable","      Averaging................"       ! Bit 8
5650    DATA "Enable","      Hardcopy In Progress......"      ! Bit 9
5660    DATA "Enable","      Service Test In Progress.."      ! Bit 10
5670    DATA "Disable","      Bit 11 Unsupported"             ! Bit 11
5680    DATA "Disable","      Bit 12 Unsupported"             ! Bit 12
5690    DATA "Disable","      Bit 13 Unsupported"             ! Bit 13
5700    DATA "Enable","      Program Running.........."       ! Bit 14
5710    DATA "Disable","      Bit 15 Unsupported"             ! Bit 15
5720 SUBEND !Decode_oper
5730 !
5740 !
5750 !*************************************************************
5760 ! Decode_meas:  Decode Measuring register.
5770 ! Description:  Decode Measuring register and Print out the
5780 !               appropriate messages.
5790 !*************************************************************
5800 !
5810 SUB Decode_meas(Reg)
5820    COM /Sys_state/ @Hp87xx,Scode
5830    DIM Message$(0:1)[120]
5840 !
5850    READ Message$(*)
5860    FOR I=0 TO 1
5870        IF BIT(Reg,I)=1 THEN
5880            PRINT Message$(I)
5890        END IF
5900    NEXT I
5910 !
5920 !  Displaying message
5930    DATA "         Meas 1 Measuring........."
5940    DATA "         Meas 2 Measuring........."
```

```
5950 SUBEND !Decode_meas
5960 !
5970 !
5980 !************************************************************
5990 ! Decode_avg:  Decode Averaging registe.
6000 ! Description: Decode Averaging register and Print out the
6010 !              appropriate messages.
6020 !************************************************************
6030 !
6040 SUB Decode_avg(Reg)
6050     COM /Sys_state/ @Hp87xx,Scode
6060     DIM Message$(0:1)[120]
6070 !
6080     READ Message$(*)
6090     FOR I=0 TO 1
6100         IF BIT(Reg,I)=1 THEN
6110             PRINT Message$(I)
6120         END IF
6130     NEXT I
6140 !
6150 !   Displaying message
6160     DATA "          Meas 1 Averaging........"
6170     DATA "          Meas 2 Averaging........"
6180 SUBEND !Decode_avg
6190 !
6200 !
6210 !************************************************************
6220 ! Decode_bias:  Decode Bias register.
6230 ! Description:  Decode Bias register and Print out the
6240 !               appropriate messages.
6250 !************************************************************
6260 !
6270 SUB Decode_bias(Meas$,Segment)
6280     COM /Sys_state/ @Hp87xx,Scode
6290     DIM Message$(0:11)[120]
6300 !
6310     READ Message$(*)
6320     SELECT Meas$
6330     CASE "Meas 1"
6340         Chan=1
6350     CASE "Meas 2"
```

```
6360         Chan=2
6370     END SELECT
6380     OUTPUT
         @Hp87xx;"CALC";VAL$(Chan);":BIAS:LIM:SEGM";VAL$(Segment);":COND?"
6390     ENTER @Hp87xx;Node
6400     FOR I=0 TO 11
6410         IF BIT(Node,I)=1 THEN
6420             PRINT Message$(I);" of ";Meas$;", Segment ";VAL$(Segment)
6430         END IF
6440     NEXT I
6450 !
6460 !   Displaying message
6470     DATA "            Current Limit Failed on  Bias 1"
6480     DATA "            Current Limit Failed on  Bias 2"
6490     DATA "            Current Limit Failed on  Bias 3"
6500     DATA "            Current Limit Failed on  Bias 4"
6510     DATA "            Current Limit Failed on  Bias 5"
6520     DATA "            Current Limit Failed on  Bias 6"
6530     DATA "            Current Limit Failed on  Bias 7"
6540     DATA "            Current Limit Failed on  Bias 8"
6550     DATA "            Current Limit Failed on  VTune "
6560     DATA "            Current Limit Failed on  VTotal"
6570     DATA "            Voltage Limit Failed on  VAux1 "
6580     DATA "            Voltage Limit Failed on  VAux2 "
6590 SUBEND !Decode_bias
6600 !
6610 !
6620 SUB Disp_err
6630 !--------------------------------------------------
6640 ! SHOW ERROR, DUMP OUT SCPI ERROR QUEUE
6650 !--------------------------------------------------
6660     COM /Sys_state/ @Hp87xx,Scode
6670 !
6680     DIM Errmsg$[400]
6690     INTEGER Errnum
6700     LOOP
6710         OUTPUT @Hp87xx;"SYST:ERR?"
6720         ENTER @Hp87xx;Errnum,Errmsg$
6730     EXIT IF Errnum=0
6740         PRINT "         ";Errnum;Errmsg$
6750     END LOOP
```

```
6760 SUBEND ! Disp_err
6770 !
6780 !
6790 !***************************************************************
6800 ! Iden_port:  Identify io port to use.
6810 !***************************************************************
6820 SUB Iden_port
6830     COM /Sys_state/ @Hp87xx,Scode
6840 !
6850     IF POS(SYSTEM$("SYSTEM ID"),"HP 87")<>0 THEN
6860         ASSIGN @Hp87xx TO 800
6870         Scode=8
6880     ELSE
6890         ASSIGN @Hp87xx TO 716
6900         Scode=7
6910     END IF
6920 !
6930 SUBEND ! Iden_port
6940 !
6950 !***************************************************************
6960 ! Setup_states:
6970 ! Note:  Insert any setup routines or statements in here.....
6980 !        This routine is execute before interrupt is enabled.
6990 !***************************************************************
7000 !
7010 SUB Setup_states
7020 !
7030 !
7040 SUBEND !Setup_states
7050 !
7060 !
7070 !***************************************************************
7080 ! Setup_srq_regs:  Set up SRQ interrupt registers.
7090 !***************************************************************
7100 SUB Setup_srq_regs
7110     COM /Sys_state/ @Hp87xx,Scode
7120 !
7130 ! Initialize interrupt registers
7140 !
7150     OUTPUT @Hp87xx;"*CLS"                  ! Clear the STATUS BYTE
    register
```

```
7160     OUTPUT @Hp87xx;"*SRE 0"               ! Clear the service request
         enable register
7170     OUTPUT @Hp87xx;"*ESE 0"               ! Clear the std event enable
         register
7180     OUTPUT @Hp87xx;"STAT:PRES"            ! Clears all other registers
7190 !
7200 ! Set up registers for interrupt on Measuring going false
7210 !
7220     OUTPUT @Hp87xx;"STAT:QUES:PTR #H06F3" ! Positive Transition
7230     OUTPUT @Hp87xx;"STAT:QUES:ENAB #H06F3"! Enable Questionable
         Register
7240     OUTPUT @Hp87xx;"STAT:OPER:PTR #HFFFF" ! Positive Transition
7250     OUTPUT @Hp87xx;"STAT:OPER:ENAB #H0600"! Enable Operational Register
7260     OUTPUT @Hp87xx;"STAT:QUES:SEGM1:PTR #H03FF"! Positive Transition
7270     OUTPUT @Hp87xx;"STAT:QUES:SEGM1:ENAB #H03FF"! Enable Segment 1
         Register
7280     OUTPUT @Hp87xx;"STAT:QUES:SEGM2:PTR #H03FF"! Positive Transition
7290     OUTPUT @Hp87xx;"STAT:QUES:SEGM2:ENAB #H03FF"! Enable Segment 2
         Register
7300     OUTPUT @Hp87xx;"STAT:QUES:SEGM3:PTR #H03FF"! Positive Transition
7310     OUTPUT @Hp87xx;"STAT:QUES:SEGM3:ENAB #H03FF"! Enable Segment 3
         Register
7320     OUTPUT @Hp87xx;"STAT:QUES:SEGM4:PTR #H03FF"! Positive Transition
7330     OUTPUT @Hp87xx;"STAT:QUES:SEGM4:ENAB #H03FF"! Enable Segment 4
         Register
7340     OUTPUT @Hp87xx;"STAT:DEV:PTR #HFF"
7350     OUTPUT @Hp87xx;"STAT:DEV:ENAB #HFF"
7360 !
7370 !
7380     OUTPUT @Hp87xx;"*CLS"                 ! Clear the STATUS BYTE
         register
7390     OUTPUT @Hp87xx;"*SRE #HFF"            ! Allow SRQ on all Registers
7400     OUTPUT @Hp87xx;"*ESE #HFF"            ! Enable standard event
         registers
7410     OUTPUT @Hp87xx;"*PSC #HFFFF"
7420 !
7430 SUBEND ! Setup_srq_regs
```

File Transfer Over HP-IB

Two example programs demonstrate how to transfer files from the analyzer's mass memory to and from mass memory of an external controller via HP-IB. Instrument states and program files may be transferred to or from the analyzers internal non-volatile memory, (`MEM:`), internal-volatile memory, (`RAM:`), and the internal 3.5" floppy disk, (`INT:`).

This can be a convenient method to archive data and programs to a central large mass storage hard drive.

To run these programs, connect an external controller to the analyzer with an HP-IB cable.

GETFILE	Transfers a file from the analyzer to an external controller.
PUTFILE	Transfers a file from an external controller to the analyzer.

GETFILE Example Program

Files are transferred from the analyzer to an external RMB controller. Run this program on your external RMB controller. The program will prompt you to specify which analyzer program to transfer, the mass storage unit (`MEM:`), internal non-volatile memory, (`RAM:`), internal volatile memory, or (`INT:`), internal 3.5" floppy disk drive and the name of the file to be created on your external controller mass storage. Transfers instrument state files or program files.

```
1000 !GETFILE
1010 !
1020 !  This program will get files from 871X specified mass storage to a host
1030 !  mass storage.  The user specifies the mass storage unit, the filename
1040 !  of the 871X and the file on the host controller to be created.
1050 !
1060 !
1070 DIM Blk$(1:10)[32000]   ! Max file size = 10 * 32000 = 320000 bytes
1080 !
1090 DIM Filename$[15],Mass$[15],Dest$[15]
1100 INTEGER Word1
1110 !
1120 COM /Sys_state/ @Hp87xx,Scode
1130 ! Identify I/O Port
1140 CALL Iden_port
1150 !
1160 BEEP
1170 Mass$="INT"
1180 Dest$="File871X"
1190 INPUT "Enter the name of the 871X file to get.",Filename$
1200 INPUT "Enter 871X Mass Storage (mem,INT,ram)",Mass$
1210 INPUT "Enter host filename (default='File871X')",Dest$
1220 DISP "READING FILE "&Mass$&":"&Filename$&" ..."
1230 OUTPUT @Hp87xx;"MMEM:TRANSFER? '"&Mass$&":"&Filename$&"'"
1240 ENTER @Hp87xx USING "#,W";Word1
1250 ENTER @Hp87xx USING "%,-K";Blk$(*)
1260 FOR I=1 TO 6
```

```
1270     Filelength=LEN(Blk$(I))+Filelength
1280 NEXT I
1290 BEEP
1300 PRINT "Length",Filelength
1310 DISP "Creating new file..."
1320 ON ERROR GOTO Save_file
1330 PURGE Dest$
1340 Save_file:      !
1350 OFF ERROR
1360 CREATE Dest$,Filelength
1370 ASSIGN @File TO Dest$;FORMAT ON
1380 OUTPUT @File;Blk$(*);
1390 ASSIGN @File TO *
1400 DISP "File "&Dest$&" created."
1410 BEEP
1420 END
1430 !
1440 !***************************************************************
1450 ! Iden_port:   Identify io port to use.
1460 ! Description: This routines sets up the I/O port address for
1470 !              the SCPI interface.  For "HP 87xx" instruments,
1480 !              the address assigned to @Hp87xx = 800 otherwise,
1490 !              716.
1500 !***************************************************************
1510 SUB Iden_port
1520     COM /Sys_state/ @Hp87xx,Scode
1530 !
1540     IF POS(SYSTEM$("SYSTEM ID"),"HP 87")<>0 THEN
1550         ASSIGN @Hp87xx TO 800
1560         Scode=8
1570     ELSE
1580         ASSIGN @Hp87xx TO 716
1590         Scode=7
1600     END IF
1610 !
1620 SUBEND !Iden_port
1630 !
```

PUTFILE Example Program

PUTFILE - Files are transferred from the RMB mass storage to the analyzer. Run this program on your external RMB controller. The program will prompt you to specify the file to transfer and where to transfer the file. BDATA or ASCII files may be transferred to the analyzer's internal non-volatile memory, (MEM:), the internal volatile memory, (RAM:), or the internal built in 3.5" floppy disk, (INT:).

```
1000 !  PUTFILE
1010 !
1020 !  This program will transfer files from RMB mass mem to the specified
1030 !  871X mass storage.  The user specifies the 871X mass storage unit,
1040 !  the 871X file to be created, file type, and file to be transferred.
1050 !
1060 !
1070 DIM A$(1:4)[32000]
1080 DIM Filename$[15],Mass$[15],Source$[15]
1090 INTEGER Word1
1100 !
1110 COM /Sys_state/ @Hp87xx,Scode
1120 ! Identify I/O Port
1130 CALL Iden_port
1140 !
1150 Bdat$="n"
1160 BEEP
1170 Mass$="INT"
1180 INPUT "Enter the name of the 871X file to create",Filename$
1190 INPUT "File type BDAT? (y,n) [n]",Bdat$
1200 INPUT "Enter the 871X Mass Storage (mem,INT,ram)",Mass$
1210 INPUT "Enter source filename",Source$
1220 DISP "READING FILE "&Source$&" ..."
1230 ASSIGN @File TO Source$;FORMAT OFF
1240 ENTER @File USING "%,-K";A$(*)
1250 ASSIGN @File TO *
1260 !PRINT A$
1270 BEEP
1280 Length=0
1290 FOR I=1 TO 4
```

```
1300     Length=LEN(A$(I))+Length
1310 NEXT I
1320 DISP "TRANSFERRING FILE = ",Length
1330 IF Bdat$="y" OR Bdat$="Y" THEN
1340     IF Length<10 THEN
1350         Blk$="1"&VAL$(Length)
1360     ELSE
1370         IF Length<100 THEN
1380             Blk$="2"&VAL$(Length)
1390         ELSE
1400             IF Length<1000 THEN
1410                 Blk$="3"&VAL$(Length)
1420             ELSE
1430                 IF Length<10000 THEN
1440                     Blk$="4"&VAL$(Length)
1450                 ELSE
1460                     IF Length<100000 THEN
1470                         Blk$="5"&VAL$(Length)
1480                     ELSE
1490                         Blk$="6"&VAL$(Length)
1500                     END IF
1510                 END IF
1520             END IF
1530         END IF
1540     END IF
1550     OUTPUT @Hp87xx;"MMEM:TRANSFER:BDAT
         '"&Mass$&":"&Filename$&"',#"&Blk$;
1560 ELSE
1570     OUTPUT @Hp87xx;"MMEM:TRANSFER '"&Mass$&":"&Filename$&"',#0";
1580 END IF
1590 OUTPUT @Hp87xx;A$(*);END
1600 DISP "871X file "&Mass$&":"&Filename$&" created."
1610 BEEP
1620 END
1630 !
```

```
1640 !**************************************************************
1650 ! Iden_port:   Identify io port to use.
1660 ! Description: This routines sets up the I/O port address for
1670 !              the SCPI interface.  For "HP 87xx" instruments,
1680 !              the address assigned to @Hp87xx = 800 otherwise,
1690 !              716.
1700 !**************************************************************
1710 SUB Iden_port
1720     COM /Sys_state/ @Hp87xx,Scode
1730 !
1740     IF POS(SYSTEM$("SYSTEM ID"),"HP 87")<>0 THEN
1750         ASSIGN @Hp87xx TO 800
1760         Scode=8
1770     ELSE
1780         ASSIGN @Hp87xx TO 716
1790         Scode=7
1800     END IF
1810 !
1820 SUBEND !Iden_port
1830 !
```

Customized Display

GRAPHICS	Using graphics and softkeys to create customized procedures. The example demonstrates the use of some of the user graphics commands including the one to erase a previously drawn line. It also demonstrates use of the softkeys and detecting a front panel keypress with the service request interrupt process.
GRAPH2	Using graphics to draw an instrument and DUT onto the display.
GETPLOT	Reading an HPGL graphics file.

GRAPHICS Example Program

This program demonstrates how to use the analyzer's user graphics commands to draw setup diagrams. It also demonstrates generating a service request in response to a keyboard interrupt. More information on user graphics commands is available in Chapter 7, "Using Graphics," and in Chapter 12, "SCPI Command Summary". Information on generating a service request and using the status reporting structure is in Chapter 5, "Using Status Registers."

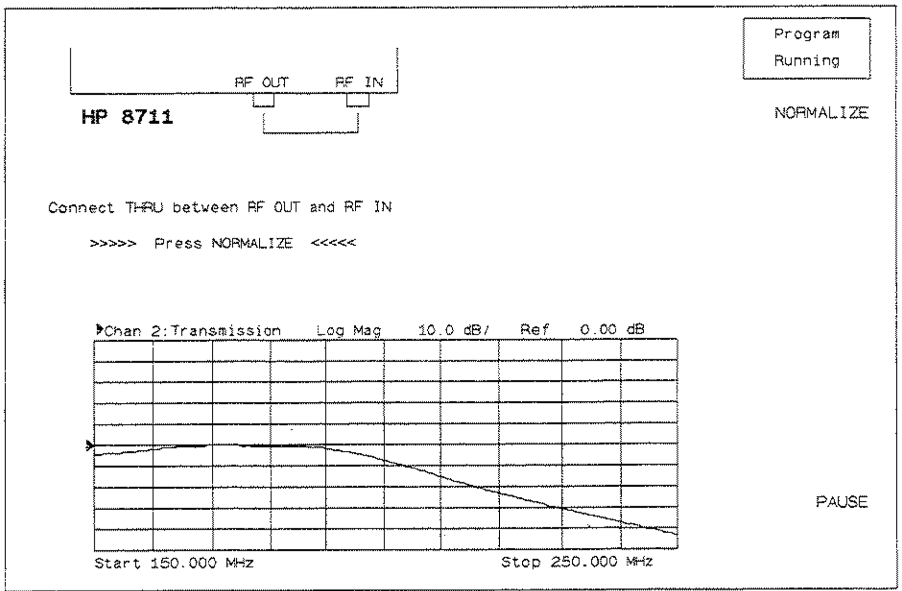
Note that this program uses the analyzer's user graphics commands. If the IBASIC option is installed, graphics may sometimes be more easily implemented using BASIC commands such as POLYGON and RECTANGLE. For further information, see the "BARCODE" program description in the *HP Instrument BASIC Users Handbook*.

Lines `170-240` draw and label a representation of an HP 8711 for a connection diagram. This example is a simple front view from the top.

Lines `250-450` draw the connection needed for a normalization. The operator is prompted to make this connection and to press a softkey on the instrument. A flashing message is used to attract attention.

NOTE

This program works properly *only* when Option 1C2, IBASIC, has been installed. Refer to program GRAPH2 if your analyzer does *not* have the IBASIC option installed.



GRAPHICS example connection diagram

Lines 460-580 perform the normalization, erase the prompts (without erasing the whole screen) and prepare for the test.

Lines 590-730 are a branching routine that handles the service request generated interrupts used by the external controller.

```
1     ! Filename:  GRAPHICS
2     !
3     ! Description: Draws a simple connection diagram
4     ! in the IBASIC window, and displays a softkey.
5     !
6     ! NOTE:  This program works properly ONLY
7     ! when option 1C2, IBASIC, has been installed.
8     ! Refer to program GRAPH2 if no IBASIC option.
9     !
```

```
10    IF POS(SYSTEM$("SYSTEM ID"),"HP 871") THEN
20      ASSIGN @Hp8711 TO 800
30      Internal=1
40      Isc=8
50    ELSE
60      ASSIGN @Hp8711 TO 716
70      Internal=0
80      Isc=7
90      ABORT 7
100     CLEAR 716
110   END IF
111   !
112   ! Allocate an IBASIC display partition
113   ! to show the graphics.
120   OUTPUT @Hp8711;"DISP:PROG UPP"
121   !
122   ! Clear the IBASIC display partition.
130   OUTPUT @Hp8711;"DISP:WIND10:GRAP:CLE"
131   !
132   ! Turn on channel 2 for measurements.  The
133   ! lower part of the display is
134   ! devoted to display of measurements.
140   OUTPUT @Hp8711;"SENS2:STAT ON;*WAI"
141   !
142   ! Take a single controlled sweep to ensure
143   ! a valid measurement using *OPC query.
150   OUTPUT @Hp8711;"ABOR;:INIT2:CONT OFF;:INIT2;*OPC?"
160   ENTER @Hp8711;Opc
161   !
162   ! Select the bright "pen" and bold font.
170   OUTPUT @Hp8711;"DISP:WIND10:GRAP:COL 1;LAB:FONT BOLD"
171   !
172   ! Draw a label reading "HP 8711C" at 45 pixels
173   ! to the right and 120 pixels above the origin.
174   ! The origin is the lower left corner of the
175   ! current graphics window (upper half).
180   OUTPUT @Hp8711;"DISP:WIND10:GRAP:MOVE 45,120
      ;LAB 'HP 8711C'"
181   !
182   ! Draw a box to represent the analyzer.
190   OUTPUT @Hp8711;"DISP:WIND10:GRAP:MOVE 30,175
```

```
     ;DRAW 30,140;DRAW 480,140;DRAW 480,175"
191  !
192  ! Draw a box to represent the REFLECTION RF OUT port.
200  OUTPUT @Hp8711;"DISP:WIND10:GRAP:MOVE 275,140
      ;DRAW 275,130;DRAW 305,130;DRAW 305,140"
201  ! Draw a box to represent the TRANSMISSION RF IN port.
210  OUTPUT @Hp8711;"DISP:WIND10:GRAP:MOVE 410,140
     ;DRAW 410,130;DRAW 440,130;DRAW 440,140"
211  ! Change the text font to small, which is the
212  ! same as that used for PRINT or DISP statements.
220  OUTPUT @Hp8711;"DISP:WIND10:GRAP:LAB:FONT SMAL"
221  !
222  ! Label the RF OUT port.
230  OUTPUT @Hp8711;"DISP:WIND10:GRAP:MOVE 250,145
     ;LAB 'RF OUT'"
231  !
232  ! Label the RF IN port.
240  OUTPUT @Hp8711;"DISP:WIND10:GRAP:MOVE 395,145
     ;LAB 'RF IN'"
241  !
250 Normalize: !
251  !
252  ! Draw a through connection between the RF OUT
253  ! and RF IN ports.
260  OUTPUT @Hp8711;"DISP:WIND10:GRAP:MOVE 290,125
     ;DRAW 290,110;DRAW 425,110;DRAW 425,125"
261  ! Prompt the operator to connect the through.
270  OUTPUT @Hp8711;"DISP:WIND10:GRAP:MOVE 1,50
     ;LAB 'Connect THRU between RF OUT and RF IN'"
280  IF Internal=1 THEN
281    ! If using the IBASIC (internal) controller,
282    ! then use the "ON KEY" method to handle
283    ! user interface.
290    ON KEY 1 LABEL "NORMALIZE" RECOVER Norm
300  ELSE
301    ! If using an external controller...
302    !
303    ! Initialize flag for checking on keyboard
304    ! interrupts.
310    Keycode=-1
311    !
```

```
312     ! Label softkey 1.
320     OUTPUT @Hp8711;"DISP:MENU:KEY1 'NORMALIZE'"
321     !
322     ! Clear the status register and event status
323     ! register.
330     OUTPUT @Hp8711;"*CLS;*ESE 0"
331     !
332     ! Preset the other status registers.
333     ! Enable the Device Status register to report
334     ! to the Status Byte on positive transition
335     ! of bit 0 (key press).  Enable the Status
336     ! Byte to generate an interrupt when the
337     ! Device Status register's summary bit
338     ! changes.
340     OUTPUT @Hp8711;"STAT:PRES;DEV:ENAB 1;*SRE 4"
341     !
342     ! Clear the key queue to ensure that previous
343     ! key presses  do not generate an interrupt.
350     OUTPUT @Hp8711;"SYST:KEY:QUE:CLE"
351     !
352     ! Set up and enable the interrupt on the HP-IB
353     ! when a service request is received.
360     ON INTR Isc,5 RECOVER Srq
370     ENABLE INTR Isc;2
380   END IF
381   !
382   ! Turn off the graphics buffer.
390   OUTPUT @Hp8711;"DISP:WIND10:GRAP:BUFF OFF"
391   !
392   ! Loop for waiting for press of the NORMALIZE key.
393   ! The two different output statements along with
394   ! the wait statements create a blinking effect.
395   ! There is not exit from this loop other than
396   ! a keyboard interrupt.
400   LOOP
410     OUTPUT @Hp8711;"DISP:WIND10:GRAP:MOVE 55,18
        ;LAB '>>>>>  Press NORMALIZE  <<<<<'"
420     WAIT .2
430     OUTPUT @Hp8711;"DISP:WIND10:GRAP:MOVE 55,18
        ;LAB '       Press NORMALIZE       '"
440     WAIT .2
```

```
450   END LOOP
451   !
460 Norm: ! Entry point to wait for a key press.
461   !
462   ! If wrong key pressed, return to Normalize.
470   IF Keycode&0 THEN GOTO Normalize
480   OFF KEY
481   !
482   ! The through should now be connected and
483   ! ready to measure.
484   !
485   ! Turn the graphics buffer back on.
490   OUTPUT @Hp8711;"DISP:WIND10:GRAP:BUFF ON"
491   !
492   ! Select the "erase" pen (pen color 0) and
493   ! erase the prompts.
500   OUTPUT @Hp8711;"DISP:WIND10:GRAP:COL 0;MOVE 55,18
      ;LAB '>>>>>  Press NORMALIZE  <<<<<'"
510   OUTPUT @Hp8711;"DISP:WIND10:GRAP:MOVE 1,50
      ;LAB 'Connect THRU between RF OUT and RF IN'"
520   OUTPUT @Hp8711;"DISP:MENU:KEY1 '          '"
521   !
522   ! Display the active data trace only.  Turn off
523   ! any previous normalization.
530   OUTPUT @Hp8711;"CALC2:MATH (IMPL)"
531   !
532   ! Take a single sweep on channel 2.
540   OUTPUT @Hp8711;"INIT2;*WAI"
541   !
542   ! Copy the new data trace into the memory array.
550   OUTPUT @Hp8711;"TRAC CH2SMEM,CH2SDATA"
551   !
552   ! Normalize; that is, display the active data
553   ! relative to the memory trace.
560   OUTPUT @Hp8711;"CALC2:MATH (IMPL/CH2SMEM)"
561   !
562   ! Display only one of the traces (the normalized
563   ! trace).
570   OUTPUT @Hp8711;"DISP:WIND2:TRAC1 ON;TRAC2 OFF"
571   !
572   ! Erase the through connect and select pen color 1 again.
```

```
580   OUTPUT @Hp8711;"DISP:WIND10:GRAP:MOVE 290,110
      ;DRAW 425,110;DRAW 425,125;COL 1"
590   STOP
600   !
610 Srq: ! This is the branching routine that handles
           service request
611        ! generated interrupts.
612   !
613   ! Do a serial poll to find out if analyzer generated the
614   ! interrupt.
620   Stb=SPOLL(@Hp8711)
621   !
622   ! Determine if the Device Status register's summary
623   ! bit (bit 2 of the Status Byte) has been set.
630   IF BINAND(Stb,4)&0 THEN
631     !
632     ! If so, then get the Device Status Register contents.
640     OUTPUT @Hp8711;"STAT:DEV:EVEN?"
650     ENTER @Hp8711;Dev_event
651     !
652     ! Check for key press...
660     IF BINAND(Dev_event,1)&0 THEN
661       ! If so, then determine which key.
670       OUTPUT @Hp8711;"SYST:KEY?"
680       ENTER @Hp8711;Keycode
690     END IF
700   END IF
701   !
702   ! Reenable the interrupt in case wrong key
703   ! was pressed.
710   ENABLE INTR Isc
720   GOTO Norm
730   END
```

GRAPH2 Example Program

This program demonstrates simple graphics and softkey handling. If the program is run from an external computer, it also demonstrates basic interrupts (SRQ) and status register handling. The program displays a hookup diagram and requests the user to connect a cable. Once the cable is connected the user is prompted to press a "NORMALIZE" key. The analyzer then performs the normalization and erases the hookup diagram.

```
1000 ! Filename:  GRAPH2
1010 !
1020 ! Description: Draws a simple connection diagram
1030 ! in the IBASIC window, and displays a softkey.
1040 !
1050 ! NOTE:  This program works properly ONLY
1060 ! when option 1C2, IBASIC, has been installed.
1070 ! Modify to use DISP:WIND1 if no IBASIC option.
1080 !
1090 !
1100 COM /Sys_state/ @Hp87xx,Scode
1110 ! Identify I/O Port
1120 CALL Iden_port
1130 !
1135   output @Hp87xx;"DISP:WIND1:GRAP:SCAL 0,1023,0,383"
1140 !
1150 ! Allocate an IBASIC display partition
1160 ! to show the graphics.
1170 OUTPUT @Hp87xx;"DISP:FORM ULOW"
1180 OUTPUT @Hp87xx;"DISP:WIND1:TRAC:GRAT:GRID OFF"
1190 !
1200 ! Clear the IBASIC display partition.
1210 OUTPUT @Hp87xx;"DISP:WIND1:GRAP:CLE"
1220 !
1230 ! Turn on channel 2 for measurements.  The
1240 ! lower part of the display is
1250 ! devoted to display of measurements.
1260 OUTPUT @Hp87xx;"SENS2:STAT ON;*WAI"
1270 !
1280 ! Take a single controlled sweep to ensure
```

```
1290 ! a valid measurement using *OPC query.
1300 OUTPUT @Hp87xx;"ABOR;:INIT2:CONT OFF;:INIT2;*OPC?"
1310 ENTER @Hp87xx;Opc
1320 !
1330 ! Select the bright "pen" and bold font.
1340 OUTPUT @Hp87xx;"DISP:WIND1:GRAP:COL 1;LAB:FONT BOLD"
1350 !
1360 ! Draw a label reading "HP 8711C" at 45 pixels
1370 ! to the right and 120 pixels above the origin.
1380 ! The origin is the lower left corner of the
1390 ! current graphics window (upper half).
1400 OUTPUT @Hp87xx;"DISP:WIND1:GRAP:MOVE 45,120;LAB 'HP 8711C'"
1410 !
1420 ! Draw a box to represent the analyzer.
1430 OUTPUT @Hp87xx;"DISP:WIND1:GRAP:MOVE 30,175;DRAW 30,140;DRAW
     480,140;DRAW 480,175"
1440 !
1450 ! Draw a box to represent the REFLECTION RF OUT port.
1460 OUTPUT @Hp87xx;"DISP:WIND1:GRAP:MOVE 275,140;DRAW 275,130;DRAW
     305,130;DRAW 305,140"
1470 ! Draw a box to represent the TRANSMISSION RF IN port.
1480 OUTPUT @Hp87xx;"DISP:WIND1:GRAP:MOVE 410,140;DRAW 410,130;DRAW
     440,130;DRAW 440,140"
1490 ! Change the text font to small, which is the
1500 ! same as that used for PRINT or DISP statements.
1510 OUTPUT @Hp87xx;"DISP:WIND1:GRAP:LAB:FONT SMAL"
1520 !
1530 ! Label the RF OUT port.
1540 OUTPUT @Hp87xx;"DISP:WIND1:GRAP:MOVE 250,145;LAB 'RF OUT'"
1550 !
1560 ! Label the RF IN port.
1570 OUTPUT @Hp87xx;"DISP:WIND1:GRAP:MOVE 380,145;LAB 'RF IN'"
1580 !
1590 Normalize: !
1600 !
1610 ! Draw a through connection between the RF OUT
1620 ! and RF IN ports.
1630 OUTPUT @Hp87xx;"DISP:WIND1:GRAP:MOVE 290,125;DRAW 290,110;DRAW
     425,110;DRAW 425,125"
1640 ! Prompt the operator to connect the through.
1650 OUTPUT @Hp87xx;"DISP:WIND1:GRAP:MOVE 1,50;LAB 'Connect THRU between RF
```

```
     OUT and RF IN'"
1660 IF Internal=1 THEN
1670 ! If using the IBASIC (internal) controller,
1680 ! then use the "ON KEY" method to handle
1690 ! user interface.
1700     ON KEY 1 LABEL "NORMALIZE" RECOVER Norm
1710 ELSE
1720 ! If using an external controller...
1730 !
1740 ! Initialize flag for checking on keyboard
1750 ! interrupts.
1760     Keycode=-1
1770 !
1780 ! Label softkey 1.
1790     OUTPUT @Hp87xx;"DISP:MENU:KEY1 'NORMALIZE'"
1800 !
1810 ! Clear the status register and event status
1820 ! register.
1830     OUTPUT @Hp87xx;"*CLS;*ESE 0"
1840 !
1850 ! Preset the other status registers.
1860 ! Enable the Device Status register to report
1870 ! to the Status Byte on positive transition
1880 ! of bit 0 (key press).  Enable the Status
1890 ! Byte to generate an interrupt when the
1900 ! Device Status register's summary bit
1910 ! changes.
1920     OUTPUT @Hp87xx;"STAT:PRES;DEV:ENAB 1;*SRE 4"
1930 !
1940 ! Clear the key queue to ensure that previous
1950 ! key presses  do not generate an interrupt.
1960     OUTPUT @Hp87xx;"SYST:KEY:QUE:CLE"
1970 !
1980 ! Set up and enable the interrupt on the HP-IB
1990 ! when a service request is received.
2000     ON INTR Scode,5 RECOVER Srq
2010     ENABLE INTR Scode;2
2020 END IF
2030 !
2040 ! Turn off the graphics buffer.
2050 OUTPUT @Hp87xx;"DISP:WIND1:GRAP:BUFF OFF"
```

```
2060 !
2070 ! Loop for waiting for press of the NORMALIZE key.
2080 ! The two different output statements along with
2090 ! the wait statements create a blinking effect.
2100 ! There is not exit from this loop other than
2110 ! a keyboard interrupt.
2120 LOOP
2130     OUTPUT @Hp87xx;"DISP:WIND1:GRAP:MOVE 55,18;LAB '>>>>>  Press
         NORMALIZE  <<<<<'"
2140     WAIT .2
2150     OUTPUT @Hp87xx;"DISP:WIND1:GRAP:MOVE 55,18;LAB '       Press
         NORMALIZE       '"
2160     WAIT .2
2170 END LOOP
2180 !
2190 Norm: ! Entry point to wait for a key press.
2200 !
2210 ! If wrong key pressed, return to Normalize.
2220 IF Keycode<>0 THEN GOTO Normalize
2230 OFF KEY
2240 !
2250 ! The through should now be connected and
2260 ! ready to measure.
2270 !
2280 ! Turn the graphics buffer back on.
2290 OUTPUT @Hp87xx;"DISP:WIND1:GRAP:BUFF ON"
2300 !
2310 ! Select the "erase" pen (pen color 0) and
2320 ! erase the prompts.
2330 OUTPUT @Hp87xx;"DISP:WIND1:GRAP:COL 0;MOVE 55,18;LAB '>>>>>  Press
     NORMALIZE  <<<<<'"
2340 OUTPUT @Hp87xx;"DISP:WIND1:GRAP:MOVE 1,50;LAB 'Connect THRU between RF
     OUT and RF IN'"
2350 OUTPUT @Hp87xx;"DISP:MENU:KEY1 '          '"
2360 !
2370 ! Display the active data trace only.  Turn off
2380 ! any previous normalization.
2390 OUTPUT @Hp87xx;"CALC2:MATH (IMPL)"
2400 !
2410 ! Take a single sweep on channel 2.
2420 OUTPUT @Hp87xx;"INIT2;*WAI"
```

```
2430 !
2440 ! Copy the new data trace into the memory array.
2450 OUTPUT @Hp87xx;"TRAC CH2SMEM,CH2SDATA"
2460 !
2470 ! Normalize; that is, display the active data
2480 ! relative to the memory trace.
2490 OUTPUT @Hp87xx;"CALC2:MATH (IMPL/CH2SMEM)"
2500 !
2510 ! Display only one of the traces (the normalized
2520 ! trace).
2530 OUTPUT @Hp87xx;"DISP:WIND2:TRAC1 ON;TRAC2 OFF"
2540 !
2550 ! Erase the through connect and select pen color 1 again.
2560 OUTPUT @Hp87xx;"DISP:WIND1:GRAP:MOVE 290,125;DRAW 290,110;DRAW
     425,110;DRAW 425,125"
2570 STOP
2580 !
2590 Srq: ! This is the branching routine that handles service request
2600 ! generated interrupts.
2610 !
2620 ! Do a serial poll to find out if analyzer generated the
2630 ! interrupt.
2640 Stb=SPOLL(@Hp87xx)
2650 !
2660 ! Determine if the Device Status register's summary
2670 ! bit (bit 2 of the Status Byte) has been set.
2680 IF BINAND(Stb,4)<>0 THEN
2690 !
2700 ! If so, then get the Device Status Register contents.
2710    OUTPUT @Hp87xx;"STAT:DEV:EVEN?"
2720    ENTER @Hp87xx;Dev_event
2730 !
2740 ! Check for key press...
2750    IF BINAND(Dev_event,1)<>0 THEN
2760 ! If so, then determine which key.
2770       OUTPUT @Hp87xx;"SYST:KEY?"
2780       ENTER @Hp87xx;Keycode
2790    END IF
2800 END IF
2810 !
2820 ! Reenable the interrupt in case wrong key.
```

```
2830 ! was pressed.
2840 ENABLE INTR Scode
2850 GOTO Norm
2860 END
2870 !
2880 !**************************************************************
2890 ! Iden_port:   Identify io port to use.
2900 ! Description: This routines sets up the I/O port address for
2910 !              the SCPI interface.  For "HP 87xx" instruments,
2920 !              the address assigned to @Hp87xx = 800 otherwise,
2930 !              716.
2940 !**************************************************************
2950 SUB Iden_port
2960     COM /Sys_state/ @Hp87xx,Scode
2970 !
2980     IF POS(SYSTEM$("SYSTEM ID"),"HP 87")<>0 THEN
2990         ASSIGN @Hp87xx TO 800
3000         Scode=8
3010     ELSE
3020         ASSIGN @Hp87xx TO 716
3030         Scode=7
3040     END IF
3050 !
3060 SUBEND !Iden_port
3070 !
```

GETPLOT Example Program

This program shows how to capture a screen plot in ".hgl" format and transfer it to the floppy drive. Although this capability now exists in firmware, this program is still useful in demonstrating file manipulation and storage. This program also allows the user to specify any filename, whereas the firmware will always choose a predefined name. This can also be used for ".pcx" format by un-commenting line 280.

Because of BASIC limitations in any single array size, a four element array is used in line 170 to store the complete file. This allows a file size up to 128,000 bytes.

Lines 320 - 340 enter the screen capture data into the array Blk$.

Lines 360- 380 determine the total number of bytes to be saved.

Lines 410 - 480 save the file. In line 420, any previous file of the same name is erased. If no file exists, this line is ignored due to the ON ERROR statement in line 410. The file is created in line 450 and the data stored in line 470.

```
100 !GETPLOT
110 !
120 !  This program will get a hardcopy screen dump in HP-GL format from
130 !  the 8711, and store it locally.
140 !  The user specifies the local filename (default = Plot871x)
150 !
160 !
170 DIM Blk$(1:4)[32000]  ! Max file size = 4 * 32000 = 128000 bytes
180 !
190 DIM Filename$[64],Dest$[64]
200 INTEGER Word1
210 !
220 COM /Sys_state/ @Hp87xx,Scode
230 ! Identify I/O Port
240 CALL Iden_port
250 !
260 BEEP
270 Filename$="DATA:screen.hgl"   ! HP-GL format
280 ! Filename$="DATA:screen.pcx"  ! PCX format
290 Dest$="Plot871x"
```

```
300 INPUT "Enter host filename (default='Plot871x')",Dest$
310 DISP "READING FILE "&Filename$&" ..."
320 OUTPUT @Hp87xx;"MMEM:TRANSFER? '"&Filename$&"'"
330 ENTER @Hp87xx USING "#,W";Word1   ! Assume indefinite block: #0 header
340 ENTER @Hp87xx USING "%,-K";Blk$(*)
350 ! Compute length of data we just ENTERed
360 FOR I=1 TO 4
370     Filelength=LEN(Blk$(I))+Filelength
380 NEXT I
390 ! Save data to local file
400 DISP "Creating new file..."
410 ON ERROR GOTO Save_file
420 PURGE Dest$
430 Save_file:     !
440 OFF ERROR
450 CREATE Dest$,Filelength
460 ASSIGN @File TO Dest$;FORMAT ON
470 OUTPUT @File;Blk$(*);
480 ASSIGN @File TO *
490 DISP "File "&Dest$&" created."
500 BEEP
510 END
520 !
530 !************************************************************
540 ! Iden_port:   Identify io port to use.
550 ! Description: This routines sets up the I/O port address for
560 !              the SCPI interface.  For "HP 87xx" instruments,
570 !              the address assigned to @Hp87xx = 800 otherwise,
580 !              716.
590 !************************************************************
```

```
600 SUB Iden_port
610     COM /Sys_state/ @Hp87xx,Scode
620 !
630     IF POS(SYSTEM$("SYSTEM ID"),"HP 87")<>0 THEN
640         ASSIGN @Hp87xx TO 800
650         Scode=8
660     ELSE
670         ASSIGN @Hp87xx TO 716
680         Scode=7
690     END IF
700 !
710 SUBEND !Iden_port
720 !
```

Annotation

USERANOT	Using user-defined annotation.
FREQBLNK	Concealing sensitive frequency information.
KEYCODES	Reading key presses and knob positions from the analyzer.

USERANOT Example Program

This program demonstrates how to use user-defined x-axis annotation. With this feature, you can set the analyzer to convert all x-axis information into a user-defined scale. In this program, the Channel 1 x-axis is modified to display antenna angle in degrees while Channel 2 x-axis displays antenna height in feet.

```
100 !RE-SAVE "USERANOT"
110 !BASIC program: An example program to draw user-defined annotation
120 !$Revision: 1.1 $
130 !$Date: 97/09/02 13:14:22 $
140 !
150 ! Demonstrate these SCPI commands:
160 !
170 !   DISPlay:ANNotation:CHANnel[1|2]:USER[:STATe] {OFF|0|ON|1}
180 !   DISPlay:ANNotation:CHANnel[1|2]:USER:LABel[:DATA] <STRING>
190 !
200 !   DISPlay:ANNotation:FREQuency[1|2]:USER[:STATe] {OFF|0|ON|1}
210 !   DISPlay:ANNotation:FREQuency[1|2]:USER:STARt #-10000~10000#
220 !   DISPlay:ANNotation:FREQuency[1|2]:USER:STOP #-10000~10000#
230 !   DISPlay:ANNotation:FREQuency[1|2]:USER:SUFFix[:DATA] <STRING>
240 !   DISPlay:ANNotation:FREQuency[1|2]:USER:LABel[:DATA] <STRING>
250 !
260 !-------------------------------------------------------------------
270 !
280 ! Determine select code (800 for IBASIC, 716 for external computer)
290 !
300   IF POS(SYSTEM$("SYSTEM ID"),"HP 87") THEN
310     ASSIGN @Hp8711 TO 800
320   ELSE
330     ASSIGN @Hp8711 TO 716
340     ABORT 7
350     CLEAR 716
360   END IF
370 !
380 ! Preset
390 OUTPUT @Hp8711;"SYST:PRES; *WAI"
400 OUTPUT @Hp8711;"SENS2:STAT ON"  ! So we can see markers
```

```
410 OUTPUT @Hp8711;"*OPC?"
420 ENTER @Hp8711;Opc
430 !
440 ! Set up channel annotation using:
450 !   DISPlay:ANNotation:CHANnel[1|2]:USER[:STATe] {OFF|0|ON|1}
460 !   DISPlay:ANNotation:CHANnel[1|2]:USER:LABel[:DATA] <STRING>
470 !
480 DISP "Setting up channel annotation..."
500 OUTPUT @Hp8711;"DISP:ANN:CHAN1:USER:STAT 1"
510 OUTPUT @Hp8711;"DISP:ANN:CHAN1:USER:LABEL 'Ch 1 Custom Annot: Signal vs.
    Antenna angle'"
520 OUTPUT @Hp8711;"DISP:ANN:CHAN2:USER:STAT 1"
530 OUTPUT @Hp8711;"DISP:ANN:CHAN2:USER:LABEL 'Ch 2 Custom Annot: Signal vs.
    Antenna height'"
540 DISP "Setting up channel annotation... Done."
550 WAIT 3
560 !
570 ! Set up frequency annotation using:
580 !   DISPlay:ANNotation:FREQuency[1|2]:USER[:STATe] {OFF|0|ON|1}
590 !   DISPlay:ANNotation:FREQuency[1|2]:USER:STARt #-10000~10000#
600 !   DISPlay:ANNotation:FREQuency[1|2]:USER:STOP #-10000~10000#
610 !   DISPlay:ANNotation:FREQuency[1|2]:USER:SUFFix[:DATA] <STRING>
620 !
630 DISP "Setting up frequency annotation..."
650 !
660 OUTPUT @Hp8711;"DISP:ANN:FREQ1:USER:LABEL 'Antenna Angle'"
670 OUTPUT @Hp8711;"DISP:ANN:FREQ1:USER:STAT 1"
680 OUTPUT @Hp8711;"DISP:ANN:FREQ1:USER:START -180.0"
690 OUTPUT @Hp8711;"DISP:ANN:FREQ1:USER:STOP   180.0"
700 OUTPUT @Hp8711;"DISP:ANN:FREQ1:USER:SUFFIX 'Deg'"
720 !
730 OUTPUT @Hp8711;"DISP:ANN:FREQ2:USER:LABEL 'Height'"
740 OUTPUT @Hp8711;"DISP:ANN:FREQ2:USER:STAT 1"
750 OUTPUT @Hp8711;"DISP:ANN:FREQ2:USER:START 5280.0"
760 OUTPUT @Hp8711;"DISP:ANN:FREQ2:USER:STOP -1760.0"
770 OUTPUT @Hp8711;"DISP:ANN:FREQ2:USER:SUFFIX 'Ft'"
780 !
790 DISP "Done.  Markers will read out using new units!"
800 OUTPUT @Hp8711;"CALC1:MARK1 ON"
810 OUTPUT @Hp8711;"CALC2:MARK1 ON"
820 !
```

```
830 LOCAL @Hp8711
840 !
850 END                                   !End of this program
```

FREQBLNK Example Program

This program demonstrates how to use user-defined x-axis annotation to conceal (or "blank") sensitive frequency information.

```
100 !RE-SAVE "FREQBLNK"
110 !BASIC program: An example program to overwrite frequency annotation
120 !$Revision: 1.2 $
130 !$Date: 97/09/02 13:17:25 $
140 !
150 ! Demonstrate using these SCPI commands to blank freq annotation.
160 !
170 !   DISPlay:ANNotation:FREQuency[1|2]:USER[:STATe] {OFF|0|ON|1}
180 !   DISPlay:ANNotation:FREQuency[1|2]:USER:STARt #-10000~10000#
190 !   DISPlay:ANNotation:FREQuency[1|2]:USER:STOP #-10000~10000#
200 !   DISPlay:ANNotation:FREQuency[1|2]:USER:SUFFix[:DATA] <STRING>
210 !   DISPlay:ANNotation:FREQuency[1|2]:USER:LABel[:DATA] <STRING>
220 !
230 !-----------------------------------------------------------------------
240 !
250 ! Determine select code (800 for IBASIC, 716 for external computer)
260 !
270   IF POS(SYSTEM$("SYSTEM ID"),"HP 87") THEN
280     ASSIGN @Hp8711 TO 800
290   ELSE
300     ASSIGN @Hp8711 TO 716
310     ABORT 7
320     CLEAR 716
330   END IF
340 !
350 ! Preset
360 OUTPUT @Hp8711;"SYST:PRES; *WAI"
370 OUTPUT @Hp8711;"SENS2:STAT ON"  ! So we can see markers
380 OUTPUT @Hp8711;"*OPC?"
390 ENTER @Hp8711;Opc
400 !
410 ! Set up frequency annotation using:
420 !   DISPlay:ANNotation:FREQuency[1|2]:USER[:STATe] {OFF|0|ON|1}
430 !   DISPlay:ANNotation:FREQuency[1|2]:USER:STARt #-10000~10000#
```

```
440 !    DISPlay:ANNotation:FREQuency[1|2]:USER:STOP #-10000~10000#
450 !    DISPlay:ANNotation:FREQuency[1|2]:USER:SUFFix[:DATA] <STRING>
460 !
470 DISP "Setting up frequency annotation..."
480 !
490 OUTPUT @Hp8711;"DISP:ANN:FREQ1:USER:LABEL 'Blank'"
500 OUTPUT @Hp8711;"DISP:ANN:FREQ1:USER:STAT 1"
510 OUTPUT @Hp8711;"DISP:ANN:FREQ1:USER:START 0.0"
520 OUTPUT @Hp8711;"DISP:ANN:FREQ1:USER:STOP  100.0"
530 OUTPUT @Hp8711;"DISP:ANN:FREQ1:USER:SUFFIX ''"
540 !
550 OUTPUT @Hp8711;"DISP:ANN:FREQ2:USER:LABEL 'Blank'"
560 OUTPUT @Hp8711;"DISP:ANN:FREQ2:USER:STAT 1"
570 OUTPUT @Hp8711;"DISP:ANN:FREQ2:USER:START 0.0"
580 OUTPUT @Hp8711;"DISP:ANN:FREQ2:USER:STOP  100.0"
590 OUTPUT @Hp8711;"DISP:ANN:FREQ2:USER:SUFFIX ''"
600 !
610 DISP "Done.  Markers will read out using new units!"
620 OUTPUT @Hp8711;"CALC1:MARK1 ON"
630 OUTPUT @Hp8711;"CALC2:MARK1 ON"
640 !
650 LOCAL @Hp8711
660 !
670 END                                        !End of this program
```

KEYCODES Example Program

This program will detect any front panel input, determine if it is from a keystroke or the knob, and display the corresponding keycode or value. Each key has a unique keycode associated with it. The knob will return either a positive or negative number depending upon direction of rotation (clockwise is positive). The program can be exited by pressing (PRESET).

Lines 1290 - 1520 are continuously repeated to look for any front panel activity.

Line 1370 - 1430 read the key type to determine if the activity came from a key press or from the knob. Also read is the value "Key_code." If the activity came from the knob, then the value "Key_code" represents how far the knob has been turned, and in which direction. If the activity came from a key stroke, the value represents the key's keycode.

Line 1510 determines if the (PRESET) key was pressed. If so, the program exits.

```
1000 ! Filename: KEYCODES
1010 ! _
1020 !
1030 !  Demonstration of how to read the analyzer's
1040 !  front panel keys and knob, as well as external
1050 !  PC keyboard, using the SCPI SYST:KEY commands.
1060 !  This program reads key presses and knob turn
1070 !  ticks and displays them on the screen.
1080 ! _
1090 !
1100 DIM Msg$[40]
1110 !
1120 !
1130 COM /Sys_state/ @Hp87xx,Scode
1140 ! Identify I/O Port
1150 CALL Iden_port
1160 !
1170 !
1180 ! Clear the key queue to ignore
1190 ! previous key presses.
```

```
1200 OUTPUT @Hp87xx;"SYST:KEY:QUE:CLE"
1210 !
1220 ! Turn on the key queue off to limit
1230 ! maximum que size, to one.
1240 OUTPUT @Hp87xx;"SYST:KEY:QUE OFF"
1250 !
1260 Msg$="'Press keys or turn knob.  PRESET ends.'"
1270 OUTPUT @Hp87xx;"DISP:ANN:MESS:DATA ";Msg$
1280 !
1290 LOOP
1300 ! Query device status condition register
1310     OUTPUT @Hp87xx;"STAT:DEV:COND?"
1320     ENTER @Hp87xx;Dev_cond
1330 !
1340 ! Check the bit that indicates a key press.
1350     IF BIT(Dev_cond,0)=1 THEN
1360 ! Read the key type first
1370         OUTPUT @Hp87xx;"SYST:KEY:TYPE?"
1380         ENTER @Hp87xx;Key_type$
1390 ! Read the key code last.
1400 ! This removes it from the queue
1410         OUTPUT @Hp87xx;"SYST:KEY?"
1420         ENTER @Hp87xx;Key_code
1430         DISP "Keycode ";Key_code;" Type ";Key_type$;
1440 ! See how many more keys are in the queue
1450         OUTPUT @Hp87xx;"SYST:KEY:QUEUE:COUNT?"
1460         ENTER @Hp87xx;Key_count
1470         DISP ". Keys in queue:";Key_count
1480     END IF
1490 !
1500 ! Stop looping if the PRESET key was pressed.
1510 EXIT IF Key_code=56 AND Key_type$="KEY"
1520 END LOOP
1530 DISP "The end."
1540 OUTPUT @Hp87xx;"DISP:ANN:MESS:DATA 'The End.'"
1550 END
1560 !
```

```
1570 !************************************************************
1580 ! Iden_port:   Identify io port to use.
1590 ! Description: This routines sets up the I/O port address for
1600 !              the SCPI interface.  For "HP 87xx" instruments,
1610 !              the address assigned to @Hp87xx = 800 otherwise,
1620 !              716.
1630 !************************************************************
1640 SUB Iden_port
1650     COM /Sys_state/ @Hp87xx,Scode
1660 !
1670     IF POS(SYSTEM$("SYSTEM ID"),"HP 87")<>0 THEN
1680         ASSIGN @Hp87xx TO 800
1690         Scode=8
1700     ELSE
1710         ASSIGN @Hp87xx TO 716
1720         Scode=7
1730     END IF
1740 !
1750 SUBEND !Iden_port
1760 !
```

Marker Functions

MKR_MATH

Marker math functions are used to calculate different parameters on a user-defined measurement trace segment. Frequency span, mean amplitude, amplitude standard deviation, and peak-to-peak amplitude are calculated with the Statistics function. Span, gain, slope and flatness are calculated with the Flatness function. Insertion loss and peak-to-peak ripple of the passband, and maximum signal amplitude in the stopband are calculated with the RF Filter Stats function. This example program steps through the marker math functions and then reads and reports the results.

MKR_MATH Example Program

```
1000 !Filename:  MKR_MATH
1010 !
1020 ! This example program demonstrates how to program marker math
1030 ! functions.  Marker Statistics, Marker Flatness, and RF Filter Stats.
1040 !
1050 ! Connect the demo filter between the RF out and RF in of the analyzer.
1060 !
1070 ! The program will step through various marker math measurements, then
1080 ! read and report the results.
1090 !
1100 !
1110 COM /Sys_state/ @Hp87xx,Scode
1120 ! Identify I/O Port
1130 CALL Iden_port
1140 !
1150 !
1160 ! Perform a system preset;
1170 OUTPUT @Hp87xx;"SYST:PRES;*WAI"
1180 !
1190 ! Set up the source frequencies for the measurement.
1200 OUTPUT @Hp87xx;"SENS1:FREQ:STAR 10 MHZ;STOP 400 MHZ;*WAI"
1210 !
1220 ! Set up the receiver for the measurement parameters
1230 ! (Transmission in this case).
1240 OUTPUT @Hp87xx;"SENS1:FUNC 'XFR:POW:RAT 2,0';DET NBAN;*WAI"
1250 !
1260 ! Configure the display so measurement
1270 ! results are easy to see.
1280 OUTPUT @Hp87xx;"DISP:WIND1:TRAC:Y:PDIV 10 DB;RLEV 0 DB;RPOS 9"
1290 !
1300 ! Reduce the distractions on the display by
1310 ! getting rid of notation that will not be
1320 ! needed in this example.
1330 OUTPUT @Hp87xx;"DISP:ANN:YAX OFF"
1340 !
1350 ! Erase the graticule grid for the same reason.
```

```
1360 OUTPUT @Hp87xx;"DISP:WIND1:TRAC:GRAT:GRID OFF"
1370 !
1380 ! Set the markers for channel 1
1390 OUTPUT @Hp87xx;"CALC1:MARK1 ON"
1400 OUTPUT @Hp87xx;"CALC1:MARK1:X 152000000.000000"
1410 OUTPUT @Hp87xx;"CALC1:MARK2 ON"
1420 OUTPUT @Hp87xx;"CALC1:MARK2:X 200000000.000000"
1430 OUTPUT @Hp87xx;"CALC1:MARK3 ON"
1440 OUTPUT @Hp87xx;"CALC1:MARK3:X 279000000.000000"
1450 OUTPUT @Hp87xx;"CALC1:MARK4 ON"
1460 OUTPUT @Hp87xx;"CALC1:MARK4:X 388000000.000000"
1470 !
1480 ! Turn on marker flatness
1490 OUTPUT @Hp87xx;"CALC1:MARK:FUNC FLATNESS"
1500 DISP "Marker Flatness"
1510 !
1520 WAIT 5
1530 OUTPUT @Hp87xx;"CALC1:MARK:FUNC:RES?"
1540 ! Read the four values:  the span, gain
1550 ! the slope, and the flatness.
1560 ENTER @Hp87xx;Span,Gain,Slope,Flatness
1570 !
1580 ! Display the results.
1590 BEEP
1600 DISP "Span ";Span
1610 !
1620 WAIT 5
1630 BEEP
1640 DISP "Gain ";Gain
1650 !
1660 WAIT 5
1670 BEEP
1680 DISP "Slope ";Slope
1690 !
1700 WAIT 5
1710 BEEP
1720 DISP "Flatness ";Flatness
1730 !
1740 WAIT 5
1750 ! Turn on marker statistics
1760 OUTPUT @Hp87xx;"CALC1:MARK:FUNC STATISTICS"
```

```
1770 DISP "Marker Statistics"
1780 !
1790 WAIT 5
1800 OUTPUT @Hp87xx;"CALC1:MARK:FUNC:RES?"
1810 ! Read the four values:  the span,
1820 ! the mean, the sdev, peak to peak.
1830 ENTER @Hp87xx;Span,Mean,Sdev,Peak
1840 !
1850 ! Display the results.
1860 BEEP
1870 DISP "Span ";Span
1880 !
1890 WAIT 5
1900 BEEP
1910 DISP "Mean ";Mean
1920 !
1930 WAIT 5
1940 BEEP
1950 DISP "Sdev ";Sdev
1960 !
1970 WAIT 5
1980 BEEP
1990 DISP "Peak ";Peak
2000 !
2010 WAIT 5
2020 ! Turn on RF Filter Stats
2030 OUTPUT @Hp87xx;"CALC1:MARK:FUNC FST"
2040 DISP "RF Filter Stats"
2050 !
2060 WAIT 5
2070 OUTPUT @Hp87xx;"CALC1:MARK:FUNC:RES?"
2080 ! Read the three values:  the loss,
2090 ! the peak to peak, and the reject
2100 ENTER @Hp87xx;Loss,Peak,Reject
2110 !
2120 ! Display the results.
2130 BEEP
2140 DISP "Loss ";Loss
2150 !
2160 WAIT 5
2170 BEEP
```

```
2180 DISP "Peak ";Peak
2190 !
2200 WAIT 5
2210 BEEP
2220 DISP "Reject ";Reject
2230 !
2240 WAIT 5
2250 DISP "Done"
2260 END
2270 !
2280 !***********************************************************
2290 ! Iden_port:   Identify io port to use.
2300 ! Description: This routines sets up the I/O port address for
2310 !              the SCPI interface.  For "HP 87xx" instruments,
2320 !              the address assigned to @Hp87xx = 800 otherwise,
2330 !              716.
2340 !***********************************************************
2350 SUB Iden_port
2360     COM /Sys_state/ @Hp87xx,Scode
2370 !
2380     IF POS(SYSTEM$("SYSTEM ID"),"HP 87")<>0 THEN
2390         ASSIGN @Hp87xx TO 800
2400         Scode=8
2410     ELSE
2420         ASSIGN @Hp87xx TO 716
2430         Scode=7
2440     END IF
2450 !
2460 SUBEND !Iden_port
2470 !
```

Marker Limit Testing

LIM_FLAT	Limit testing can be performed on the flatness of a user-defined measurement trace segment. This example program sets various flatness limits, then queries the status to determine if the limit test passes or fails.
LIM_PEAK	Limit testing can be performed on the peak-to-peak ripple of a user-defined measurement trace segment. This example program sets various peak-to-peak limits, then queries the status to determine if the limit test passes or fails.
LIM_MEAN	Limit testing can be performed on the mean amplitude of a user-defined measurement trace segment. This example program sets various mean limits, then queries the status to determine if the limit test passes or fails.

LIM_FLAT Example Program

```
10   !Filename:  LIM_FLAT
20   !
30   ! This example program demonstrates how to test for a marker
40   ! flatness limit.
50   !
60   ! Connect the demo filter to the analyzer RF out and RF in.
70   ! The analyzer will set-up a transmission measurement.
80   !
90   ! The program will set various flatness limits, then query the
100  ! status to determine if the specification PASSES or FAILS.
110  !
120  !
130   IF POS(SYSTEM$("SYSTEM ID"),"HP 871") THEN
140     ASSIGN @Hp8711 TO 800
150   ELSE
160     ASSIGN @Hp8711 TO 716
170     ABORT 7
180     CLEAR 716
190   END IF
200  !
210  ! Perform a system preset; this clears the limit table.
220   OUTPUT @Hp8711;"SYST:PRES;*WAI"
230  !
240  ! Set up the source frequencies for the measurement.
250   OUTPUT @Hp8711;"SENS1:FREQ:STAR 10 MHZ;STOP 400 MHZ;*WAI"
260  !
270  ! Set up the receiver for the measurement parameters
280  ! (Transmission in this case).
290   OUTPUT @Hp8711;"SENS1:FUNC 'XFR:POW:RAT 2,0';DET NBAN;*WAI"
300  !
310  ! Configure the display so measurement
320  ! results are easy to see.
330   OUTPUT @Hp8711;"DISP:WIND1:TRAC:Y:PDIV 10 DB;RLEV 0 DB;RPOS 9"
340  !
350  ! Reduce the distractions on the display by
360  ! getting rid of notation that will not be
```

```
370 ! needed in this example.
380  OUTPUT @Hp8711;"DISP:ANN:YAX OFF"
390 !
400 ! Erase the graticule grid for the same reason.
410  OUTPUT @Hp8711;"DISP:WIND1:TRAC:GRAT:GRID OFF"
420 !
430 ! Set the markers for channel 1
440  OUTPUT @Hp8711;"CALC1:MARK1 ON"
450  OUTPUT @Hp8711;"CALC1:MARK1:X 152000000.000000"
460  OUTPUT @Hp8711;"CALC1:MARK2 ON"
470  OUTPUT @Hp8711;"CALC1:MARK2:X 200000000.000000"
480 !
490 ! Turn on marker flatness
500  OUTPUT @Hp8711;"CALC1:MARK:FUNC FLATNESS"
510 !
520  OUTPUT @Hp8711;"CALC1:MARK2 ON"
530  OUTPUT @Hp8711;"CALC1:LIM:DISP ON"
540  OUTPUT @Hp8711;"CALC1:LIM:MARK:FLATNESS ON"
550 !
560 ! Turn on the pass/fail testing; watch the
570 ! analyzer's display for the pass/fail indicator.
580  OUTPUT @Hp8711;"CALC1:LIM:STAT ON"
590 !
600 !  Set sweep hold mode
610  OUTPUT @Hp8711;"ABOR;:INIT1:CONT OFF;:INIT1;*WAI"
620 !
630 ! Send an operation complete query to ensure that
640 ! all overlapped commands have been executed.
650  OUTPUT @Hp8711;"*OPC?"
660 !
670 ! Wait for the reply.
680  ENTER @Hp8711;Opc
690 !
700 ! Turn on a limit to be tested
710  FOR Flatness=0. TO 3 STEP .1
720    DISP "Flatness limit test =",VAL$(Flatness)&" dB"
730    OUTPUT @Hp8711;"CALC1:LIM:MARK:FLAT:MAX "&VAL$(Flatness)
740 !
750 ! Take a controlled sweep to ensure that
760 ! there is real data present for the limit test.
770    OUTPUT @Hp8711;"INIT1;*OPC?"
```

```
780     ENTER @Hp8711;Opc
790   !
800   ! Query the limit fail condition register to see
810   ! if there is a failure.
820     OUTPUT @Hp8711;"STAT:QUES:LIM:COND?"
830   !
840   ! Read the register's contents.
850     ENTER @Hp8711;Fail_flag
860   !
870   ! Bit 0 is the test result for channel 1 while
880   ! Bit 1 is the results for channel 2 limit testing.
890   ! Bit 2 is the result for channel 1 mkr limit testing.
900   ! Bit 3 is the result for channel 2 mkr limit testing.
910     IF BIT(Fail_flag,2)=1 THEN
920   ! This limit test failed
930     ELSE
940       DISP "Flatness passed at "&VAL$(Flatness)&" dB"
950       BEEP
960       GOTO Done
970     END IF
980   !
990    NEXT Flatness
1000 Done:OUTPUT @Hp8711;"INIT:CONT ON;*WAI"
1010   END
```

LIM_PEAK Example Program

```
1000 !Filename: LIM_PEAK
1010 !
1020 ! This example program demonstrates how to test for a marker
1030 ! statistics peak to peak ripple limit.
1040 !
1050 ! Connect the demo filter to the analyzer RF out and RF in.
1060 ! The analyzer will set-up a transmission measurement.
1070 !
1080 ! The program will set various statistics peak to peak limits, then
1090 ! query the status to determine if the specification PASSES or FAILS.
1100 !
1110 !
1120 !
1130 COM /Sys_state/ @Hp87xx,Scode
1140 ! Identify I/O Port
1150 CALL Iden_port
1160 !
1170 !
1180 ! Perform a system preset; this clears the limit table.
1190 OUTPUT @Hp87xx;"SYST:PRES;*WAI"
1200 !
1210 ! Set up the source frequencies for the measurement.
1220 OUTPUT @Hp87xx;"SENS1:FREQ:STAR 10 MHZ;STOP 400 MHZ;*WAI"
1230 !
1240 ! Set up the receiver for the measurement parameters
1250 ! (Transmission in this case).
1260 OUTPUT @Hp87xx;"SENS1:FUNC 'XFR:POW:RAT 2,0';DET NBAN;*WAI"
1270 !
1280 ! Configure the display so measurement
1290 ! results are easy to see.
1300 OUTPUT @Hp87xx;"DISP:WIND1:TRAC:Y:PDIV 10 DB;RLEV 0 DB;RPOS 9"
1310 !
1320 ! Reduce the distractions on the display by
1330 ! getting rid of notation that will not be
1340 ! needed in this example.
1350 OUTPUT @Hp87xx;"DISP:ANN:YAX OFF"
```

```
1360 !
1370 ! Erase the graticule grid for the same reason.
1380 OUTPUT @Hp87xx;"DISP:WIND1:TRAC:GRAT:GRID OFF"
1390 !
1400 ! Set the markers for channel 1
1410 OUTPUT @Hp87xx;"CALC1:MARK1 ON"
1420 OUTPUT @Hp87xx;"CALC1:MARK1:X 152000000.000000"
1430 OUTPUT @Hp87xx;"CALC1:MARK2 ON"
1440 OUTPUT @Hp87xx;"CALC1:MARK2:X 200000000.000000"
1450 !
1460 ! Turn on marker statistics
1470 OUTPUT @Hp87xx;"CALC1:MARK:FUNC STATISTICS"
1480 !
1490 OUTPUT @Hp87xx;"CALC1:MARK2 ON"
1500 OUTPUT @Hp87xx;"CALC1:LIM:DISP ON"
1510 OUTPUT @Hp87xx;"CALC1:LIM:MARK:STAT:PEAK ON"
1520 !
1530 ! Turn on the pass/fail testing; watch the
1540 ! analyzer's display for the pass/fail indicator.
1550 OUTPUT @Hp87xx;"CALC1:LIM:STAT ON"
1560 !
1570 !  Set sweep hold mode
1580 OUTPUT @Hp87xx;"ABOR;:INIT1:CONT OFF;:INIT1;*WAI"
1590 !
1600 ! Send an operation complete query to ensure that
1610 ! all overlapped commands have been executed.
1620 OUTPUT @Hp87xx;"*OPC?"
1630 !
1640 ! Wait for the reply.
1650 ENTER @Hp87xx;Opc
1660 !
1670 ! Turn on a limit to be tested
1680 FOR Peak_limit=0. TO 3 STEP .1
1690     DISP "Peak limit test =",VAL$(Peak_limit)&" dB"
1700     OUTPUT @Hp87xx;"CALC1:LIM:MARK:STAT:PEAK:MAX "&VAL$(Peak_limit)
1710 !
1720 ! Send an operation complete query to ensure that
1730 ! all overlapped commands have been executed.
1740     OUTPUT @Hp87xx;"*OPC?"
1750 !
1760 ! Wait for the reply.
```

```
1770     ENTER @Hp87xx;Opc
1780 !
1790 ! Take a controlled sweep to ensure that
1800 ! there is real data present for the limit test.
1810     OUTPUT @Hp87xx;"INIT1;*OPC?"
1820     ENTER @Hp87xx;Opc
1830 !
1840 ! Query the limit fail condition register to see
1850 ! if there is a failure.
1860     OUTPUT @Hp87xx;"STAT:QUES:LIM:COND?"
1870 !
1880 ! Read the register's contents.
1890     ENTER @Hp87xx;Fail_flag
1900 !
1910 ! Bit 0 is the test result for channel 1 while
1920 ! Bit 1 is the results for channel 2 limit testing.
1930 ! Bit 2 is the result for channel 1 mkr limit testing.
1940 ! Bit 3 is the result for channel 2 mkr limit testing.
1950     IF BIT(Fail_flag,2)=1 THEN
1960 ! This limit test failed
1970     ELSE
1980         DISP "Passed at "&VAL$(Peak_limit)&" dB"
1990         BEEP
2000         GOTO Done
2010     END IF
2020 !
2030 NEXT Peak_limit
2040 Done:OUTPUT @Hp87xx;"INIT:CONT ON;*WAI"
2050 END
2060 !
2070 !************************************************************
2080 ! Iden_port:   Identify io port to use.
2090 ! Description: This routines sets up the I/O port address for
2100 !              the SCPI interface.  For "HP 87xx" instruments,
2110 !              the address assigned to @Hp87xx = 800 otherwise,
2120 !              716.
2130 !************************************************************
```

```
2140 SUB Iden_port
2150     COM /Sys_state/ @Hp87xx,Scode
2160 !
2170     IF POS(SYSTEM$("SYSTEM ID"),"HP 87")<>0 THEN
2180         ASSIGN @Hp87xx TO 800
2190         Scode=8
2200     ELSE
2210         ASSIGN @Hp87xx TO 716
2220         Scode=7
2230     END IF
2240 !
2250 SUBEND !Iden_port
2260 !
```

LIM_MEAN Example Program

```
1000 !Filename:  LIM_MEAN
1010 !
1020 ! This example program demonstrates how to test for a marker
1030 ! statistics mean limit.
1040 !
1050 ! Connect the demo filter to the analyzer RF out and RF in.
1060 ! The analyzer will set-up a transmission measurement.
1070 !
1080 ! The program will set various statistics mean limits, then query
1090 ! the status to determine if the specification PASSES or FAILS.
1100 !
1110 !
1120 !
1130 COM /Sys_state/ @Hp87xx,Scode
1140 ! Identify I/O Port
1150 CALL Iden_port
1160 !
1170 !
1180 ! Perform a system preset; this clears the limit table.
1190 OUTPUT @Hp87xx;"SYST:PRES;*WAI"
1200 !
1210 ! Set up the source frequencies for the measurement.
1220 OUTPUT @Hp87xx;"SENS1:FREQ:STAR 10 MHZ;STOP 400 MHZ;*WAI"
1230 !
1240 ! Set up the receiver for the measurement parameters
1250 ! (Transmission in this case).
1260 OUTPUT @Hp87xx;"SENS1:FUNC 'XFR:POW:RAT 2,0';DET NBAN;*WAI"
1270 !
1280 ! Configure the display so measurement
1290 ! results are easy to see.
1300 OUTPUT @Hp87xx;"DISP:WIND1:TRAC:Y:PDIV 10 DB;RLEV 0 DB;RPOS 9"
1310 !
1320 ! Reduce the distractions on the display by
1330 ! getting rid of notation that will not be
1340 ! needed in this example.
1350 OUTPUT @Hp87xx;"DISP:ANN:YAX OFF"
```

```
1360 !
1370 ! Erase the graticule grid for the same reason.
1380 OUTPUT @Hp87xx;"DISP:WIND1:TRAC:GRAT:GRID OFF"
1390 !
1400 ! Set the markers for channel 1
1410 OUTPUT @Hp87xx;"CALC1:MARK1 ON"
1420 OUTPUT @Hp87xx;"CALC1:MARK1:X 152000000.000000"
1430 OUTPUT @Hp87xx;"CALC1:MARK2 ON"
1440 OUTPUT @Hp87xx;"CALC1:MARK2:X 200000000.000000"
1450 !
1460 ! Turn on marker statistics
1470 OUTPUT @Hp87xx;"CALC1:MARK:FUNC STATISTICS"
1480 !
1490 OUTPUT @Hp87xx;"CALC1:MARK2 ON"
1500 OUTPUT @Hp87xx;"CALC1:LIM:DISP ON"
1510 OUTPUT @Hp87xx;"CALC1:LIM:MARK:STAT:MEAN ON"
1520 !
1530 ! Turn on the pass/fail testing; watch the
1540 ! analyzer's display for the pass/fail indicator.
1550 OUTPUT @Hp87xx;"CALC1:LIM:STAT ON"
1560 !
1570 !  Set sweep hold mode
1580 OUTPUT @Hp87xx;"ABOR;:INIT1:CONT OFF;:INIT1;*WAI"
1590 !
1600 ! Send an operation complete query to ensure that
1610 ! all overlapped commands have been executed.
1620 OUTPUT @Hp87xx;"*OPC?"
1630 !
1640 ! Wait for the reply.
1650 ENTER @Hp87xx;Opc
1660 !
1670 ! Turn on a limit to be tested
1680 FOR Mean_limit=0. TO -5 STEP -.1
1690      DISP "Mean limit test =",VAL$(Mean_limit)&" dB"
1700      OUTPUT @Hp87xx;"CALC1:LIM:MARK:STAT:MEAN:MIN "&VAL$(Mean_limit)
1710 !
1720 ! Send an operation complete query to ensure that
1730 ! all overlapped commands have been executed.
1740      OUTPUT @Hp87xx;"*OPC?"
1750 !
1760 ! Wait for the reply.
```

```
1770      ENTER @Hp87xx;Opc
1780 !
1790 ! Take a controlled sweep to ensure that
1800 ! there is real data present for the limit test.
1810      OUTPUT @Hp87xx;"INIT1;*OPC?"
1820      ENTER @Hp87xx;Opc
1830 !
1840 ! Query the limit fail condition register to see
1850 ! if there is a failure.
1860      OUTPUT @Hp87xx;"STAT:QUES:LIM:COND?"
1870 !
1880 ! Read the register's contents.
1890      ENTER @Hp87xx;Fail_flag
1900 !
1910 ! Bit 0 is the test result for channel 1 while
1920 ! Bit 1 is the results for channel 2 limit testing.
1930 ! Bit 2 is the result for channel 1 mkr limit testing.
1940 ! Bit 3 is the result for channel 2 mkr limit testing.
1950      IF BIT(Fail_flag,2)=1 THEN
1960 ! This limit test failed
1970      ELSE
1980          DISP "Passed at "&VAL$(Mean_limit)&" dB"
1990          BEEP
2000          GOTO Done
2010      END IF
2020 !
2030 NEXT Mean_limit
2040 Done:OUTPUT @Hp87xx;"INIT:CONT ON;*WAI"
2050 END
2060 !
2070 !***************************************************************
2080 ! Iden_port:   Identify io port to use.
2090 ! Description: This routines sets up the I/O port address for
2100 !              the SCPI interface.  For "HP 87xx" instruments,
2110 !              the address assigned to @Hp87xx = 800 otherwise,
2120 !              716.
2130 !***************************************************************
```

```
2140 SUB Iden_port
2150     COM /Sys_state/ @Hp87xx,Scode
2160 !
2170     IF POS(SYSTEM$("SYSTEM ID"),"HP 87")<>0 THEN
2180         ASSIGN @Hp87xx TO 800
2190         Scode=8
2200     ELSE
2210         ASSIGN @Hp87xx TO 716
2220         Scode=7
2230     END IF
2240 !
2250 SUBEND !Iden_port
2260 !
```

SRL Measurements (Option 100 only)

MEAS_SRL	This programs shows the effects of various connector modeling parameters on an SRL measurement.
SRL_SRQ	This program initiates an SRL cable scan and sets up the analyzer to send an SRQ interrupt when the scan is completed.

MEAS_SRL Example Program

```
1000 ! Filename: MEAS_SRL (option 100 only)
1010 !
1020 ! This program is designed to show the effects of the various
1030 ! connector modeling on an SRL measurement.
1040 !
1050 ! For this measurement:  Users can change the following
1060 ! parameters.  Each parameter can be adjusted either
1070 ! manually or can be determined automatically by the
1080 ! analyzer.
1090 !
1100 ! To measure SRL of a cable, connect a long cable terminated
1110 ! with a load standard.  (50 or 75 ohm).
1120 ! The program steps through various settings.
1130 !
1140 ! Cable Z           -  Cable impedance
1150 ! Cable Zstop       -  The max freq in which Z average is measrued
1160 ! Connector C       -  Connector Capacitance
1170 ! Connector Length  -  Connector Length
1180 !
1190 ! After several values have been tried, the command is sent to
1200 ! measure the connector and automatically determine the optimum
1210 ! connector model values.
1220 !
1230 !
1240 COM /Sys_state/ @Hp87xx,Scode
1250 ! Identify I/O Port
1260 CALL Iden_port
1270 !
1280 OUTPUT @Hp87xx;"SYST:PRES; *OPC?"
1290 ENTER @Hp87xx;Opc
1300 !
1310 ! Select the SRL measurement on channel 1
1320 OUTPUT @Hp87xx;"SENS1:STAT ON; *WAI"
1330 OUTPUT @Hp87xx;"SENS1:FUNC 'SRL 1,0';DET NBAN; *WAI"
1340 !
1350 ! Sweep Hold mode
```

```
1360 OUTPUT @Hp87xx;"ABOR;:INIT1:CONT OFF;*OPC?"
1370 ENTER @Hp87xx;Opc
1380 !
1390 ! Take a sweep
1400 OUTPUT @Hp87xx;"ABOR;:INIT1:CONT OFF;:INIT1;*WAI"
1410 BEEP
1420 !
1430 Clear_disp
1440 Disp_mess("SRL connector model test...")
1450 WAIT 5
1460 !
1470 CALL Meas_srl(0.,0,0.,2.10E+8)
1480 ! Change srl parameters and re-measure
1490 Clear_disp
1500 Disp_mess("Setting default settings...")
1510 CALL Meas_srl(0.,0,0.,2.10E+8)
1520 WAIT 4
1530 !
1540 Clear_disp
1550 Disp_mess("Setting C = -1 pF...")
1560 CALL Meas_srl(0.,-1.E-12,0.,2.10E+8)
1570 WAIT 4
1580 !
1590 Clear_disp
1600 Disp_mess("Setting L = 50 mm...")
1610 CALL Meas_srl(0.,-1.E-12,.050,2.10E+8)
1620 WAIT 4
1630 !
1640 Clear_disp
1650 Disp_mess("Setting manual Z to 76 Ohm...")
1660 CALL Meas_srl(76.,-1.E-12,.050,2.10E+8)
1670 WAIT 4
1680 !
1690 Clear_disp
1700 Disp_mess("Auto Z with z_cutoff = 1E9 Hz")
1710 CALL Meas_srl(0.,-1.E-12,.050,1.E+9)
1720 WAIT 4
1730 !
1740 Clear_disp
1750 Disp_mess("Optimize connector model...")
1760 CALL Meas_srl(-1.,-1.E-12,0.,2.1E+6)
```

```
1770 WAIT 4
1780 !
1790 Clear_disp
1800 BEEP
1810 END
1820 !
1830 SUB Meas_srl(REAL Z,REAL Cap,REAL Length,REAL Zstop)
1840    COM /Sys_state/ @Hp87xx,Scode
1850    DIM Msg$[80]
1860    IF Z>=0 THEN
1870        WAIT 1
1880        OUTPUT @Hp87xx;"SENS1:CORR:LENG:CONN "&VAL$(Length)
1890        OUTPUT @Hp87xx;"SENS1:CORR:CAP:CONN "&VAL$(Cap)
1900        OUTPUT @Hp87xx;"SENS:FREQ:ZST "&VAL$(Zstop)
1910        IF Z>1. THEN
1920 ! Set manual impedance mode
1930            OUTPUT @Hp87xx;"SENS1:FUNC:SRL:MODE MANUAL"
1940            OUTPUT @Hp87xx;"SENS1:FUNC:SRL:IMP "&VAL$(Z)
1950        ELSE
1960 ! Automatically measure the impedance
1970            OUTPUT @Hp87xx;"SENS1:FUNC:SRL:MODE AUTO"
1980        END IF
1990 ! Take a sweep
2000        OUTPUT @Hp87xx;"ABOR;:INIT1:CONT OFF;:INIT1;*OPC?"
2010        ENTER @Hp87xx;Opc
2020    ELSE
2030 ! Automatically determine the srl connector model
2040        OUTPUT @Hp87xx;"SENS1:CORR:MODEL:CONN;*OPC?"
2050        ENTER @Hp87xx;Opc
2060    END IF
2070    BEEP
2080    OUTPUT @Hp87xx;"SENS1:CORR:LENG:CONN?"
2090    ENTER @Hp87xx;L
2100    OUTPUT @Hp87xx;"SENS1:CORR:CAP:CONN?"
2110    ENTER @Hp87xx;C
2120 ! Read the impedance.
2130 ! In AUTOMATIC_Z mode, the returned impedance is the measured Z.
2140 ! In MANUAL Z mode, the returned impedance is the manually entered Z.
2150    OUTPUT @Hp87xx;"SENS1:FUNC:SRL:IMP?"
2160    ENTER @Hp87xx;Zmeas
2170    Zmeas=DROUND(Zmeas,3)
```

```
2180     Cnew=DROUND(C,3)
2190     Lnew=DROUND(L,3)
2200     Msg$="C="&VAL$(Cnew)&" F,  L="&VAL$(Lnew)&" m, Z="&VAL$(Zmeas)&" Ohm"
2210     Clear_disp
2220     Disp_mess(Msg$)
2230 SUBEND
2240 !
2250 SUB Disp_mess(Message$)
2260     COM /Sys_state/ @Hp87xx,Scode
2270     OUTPUT @Hp87xx;"DISP:ANN:MESS:DATA '"&Message$&"'"
2280 SUBEND
2290 !
2300 SUB Clear_disp
2310     COM /Sys_state/ @Hp87xx,Scode
2320     DIM Command$[40]
2330     OUTPUT @Hp87xx;"DISP:ANN:MESS:CLE"
2340 SUBEND
2350 !
2360 !**************************************************************
2370 ! Iden_port:   Identify io port to use.
2380 ! Description: This routines sets up the I/O port address for
2390 !              the SCPI interface.  For "HP 87xx" instruments,
2400 !              the address assigned to @Hp87xx = 800 otherwise,
2410 !              716.
2420 !**************************************************************
2430 SUB Iden_port
2440     COM /Sys_state/ @Hp87xx,Scode
2450 !
2460     IF POS(SYSTEM$("SYSTEM ID"),"HP 87")<>0 THEN
2470         ASSIGN @Hp87xx TO 800
2480         Scode=8
2490     ELSE
2500         ASSIGN @Hp87xx TO 716
2510         Scode=7
2520     END IF
2530 !
2540 SUBEND !Iden_port
2550 !
```

SRL_SRQ Example Program

```
1000 !Filename:  SRL_SRQ (option 100 only)
1010 !
1020 ! Description:
1030 !
1040 ! This example program demonstrates how to initiate an SRL
1050 ! cable scan.  The instrument is set-up to send a
1060 ! SRQ interrupt when the scan has been completed.
1070 !
1080 ! Connect the cable to be tested to the RF out port on the
1090 ! analyzer.
1100 !
1110 ! Set an SRQ to occur when the SRL scan is complete.
1120 !
1130 !
1140 COM /Sys_state/ @Hp87xx,Scode
1150 ! Identify I/O Port
1160 CALL Iden_port
1170 !
1180 !
1190 ! Preset the instrument
1200 OUTPUT @Hp87xx;"SYST:PRES;*OPC?"
1210 ENTER @Hp87xx;Opc
1220 !
1230 ! Turn on SRL measurement
1240 OUTPUT @Hp87xx;"SENS1:FUNC 'SRL 1,0';DET NBAN;*OPC?"
1250 ENTER @Hp87xx;Opc
1260 !
1270 ! Clear status registers.
1280 OUTPUT @Hp87xx;"*CLS"
1290 !
1300 ! Clear the Service Request Enable register.
1310 OUTPUT @Hp87xx;"*SRE 0"
1320 !
1330 ! Clear the Standard Event Status Enable register.
1340 OUTPUT @Hp87xx;"*ESE 0"
1350 !
```

```
1360 ! Preset the remaining status registers.
1370 OUTPUT @Hp87xx;"STAT:PRES"
1380 !
1390 ! Set operation status register to report
1400 ! to the status byte on NEGATIVE transition
1410 ! the srl bit.
1420 OUTPUT @Hp87xx;"STAT:OPER:ENAB 16"
1430 OUTPUT @Hp87xx;"STAT:OPER:MEAS:PTR #H0000"
1440 OUTPUT @Hp87xx;"STAT:OPER:MEAS:NTR #HFFFF"
1450 !
1460 ! Set measuring status register to report to
1470 ! operational status register on NEGATIVE transition
1480 ! of the srl scan done bits.  The NEGATIVE
1490 ! transition needs to be detected because the
1500 ! srl= bit 3 is set to 1 while the analyzer
1510 ! is sweeping on channel 1  When the bit
1520 ! goes back to 0, the srl scan is done.
1530 OUTPUT @Hp87xx;"STAT:OPER:MEAS:ENAB 4"
1540 !
1550 ! Enable the operational status bit in the status
1560 ! byte to generate an SRQ.
1570 OUTPUT @Hp87xx;"*SRE 128"
1580 !
1590 ! On an interrupt from HP-IB "Scode" (Interface
1600 ! Select Code) SRQ bit (2), branch to the interrupt
1610 ! service routine "Srq_handler".
1620 ON INTR Scode,2 GOSUB Srq_handler
1630 !
1640 ! Initialize flag indicating when srl scan done
1650 ! to 0.  Then loop continuously until the
1660 ! interrupt is detected, and the interrupt
1670 ! service routine acknowledges the
1680 ! interrupt and sets the flag to 1.
1690 !
1700 Srl_done=0
1710 ! Now enable the interrupt on SRQ (Service Request).
1720 ENABLE INTR Scode;2
1730 !
1740 ! Initiate the SRL sweep
1750 OUTPUT @Hp87xx;"SENS1:FUNC:SRL:SCAN;*WAI"
1760 !
```

```
1770 DISP "Waiting for SRQ on srl scan done.";
1780 LOOP
1790     DISP ".";
1800     WAIT 1! Slow down dots
1810 EXIT IF Srl_done=1
1820 END LOOP
1830 !
1840 ! Display desired completion message.
1850 DISP
1860 DISP "Got SRQ.  SRL Scan!"
1870 STOP
1880 !
1890 Srq_handler:  ! Interrupt Service Routine
1900 !
1910 ! Determine that the analyzer was actually
1920 ! the instrument that generated the
1930 ! interrupt.
1940 Stb=SPOLL(@Hp87xx)
1950 !
1960 ! Determine if the operation status register
1970 ! caused the interrupt by looking at bit 7
1980 ! of the result of the serial poll.
1990 IF BINAND(Stb,128)<>0 THEN
2000 !
2010 ! Read the operational status event register.
2020     OUTPUT @Hp87xx;"STAT:OPER:EVEN?"
2030     ENTER @Hp87xx;Op_event
2040 !
2050 ! Determine if the srl status register
2060 ! bit 4 is set.
2070     IF BINAND(Op_event,16)<>0 THEN
2080 !
2090 ! If so, then set flag indicating
2100 ! srl scan done.
2110         Srl_done=1
2120     END IF
2130 END IF
2140 RETURN
2150 END
2160 !
```

```
2170 !**************************************************************
2180 ! Iden_port:   Identify io port to use.
2190 ! Description: This routines sets up the I/O port address for
2200 !              the SCPI interface.  For "HP 87xx" instruments,
2210 !              the address assigned to @Hp87xx = 800 otherwise,
2220 !              716.
2230 !**************************************************************
2240 SUB Iden_port
2250     COM /Sys_state/ @Hp87xx,Scode
2260 !
2270     IF POS(SYSTEM$("SYSTEM ID"),"HP 87")<>0 THEN
2280         ASSIGN @Hp87xx TO 800
2290         Scode=8
2300     ELSE
2310         ASSIGN @Hp87xx TO 716
2320         Scode=7
2330     END IF
2340 !
2350 SUBEND !Iden_port
2360 !
```

Fault Location Measurements (Option 100 only)

FAULT	This programs shows the effects of various fault location frequency modes on a cable measurement.
USR_FLOC	This program shows how to simplify fault location measurements by using the `User BEGIN` key. (You must have Option 1C2, IBASIC, to use the `User BEGIN` key.)

FAULT Example Program

```
1000 ! Filename: FAULT (Option 100 only)
1010 !
1020 ! This program is designed to show the affects of the various
1030 ! fault location frequency modes on a cable measurement.
1040 !
1050 ! Connect a 50 m. (150ft) cable to the RF out of the analyzer.
1060 ! (if available).
1070 !
1080 ! The program steps through various settings.
1090 !
1100 ! Set Feet/Meters
1110 ! Start Distance   0   meters
1120 ! Stop Distance    100 meters
1130 ! Low Pass mode
1140 ! Band Pass mode   CF  = 600 MHz
1150 ! Band Pass mode   CF  = 900 MHz
1160 ! Low Pass mode
1170 ! Cable Loss
1180 ! Cable Velocity Factor
1190 !
1200 ! The commands which cause changes to frequency settings will
1210 ! cause the analyzer to automatically display a caution message
1220 ! to verify Cable Loss and Velocity Factor.
1230 !
1240 !
1250 COM /Sys_state/ @Hp87xx,Scode
1260 ! Identify I/O Port
1270 CALL Iden_port
1280 !
1290 ! Preset the analyzer
1300 OUTPUT @Hp87xx;"SYST:PRES; *OPC?"
1310 ENTER @Hp87xx;Opc
1320 !
1330 ! Enable fault location measurment on channel 1
1340 OUTPUT @Hp87xx;"SENS1:STAT ON; *WAI"
1350 OUTPUT @Hp87xx;"SENS1:FUNC 'FLOC 1,0';DET NBAN; *OPC?"
```

```
1360 ENTER @Hp87xx;Opc
1370 WAIT 2
1380 !
1390 !  Autoscale the fault measurment
1400 OUTPUT @Hp87xx;"DISP:WIND1:TRAC:Y:AUTO ONCE"
1410 !
1420 Clear_disp
1430 Disp_mess("Fault Location Demo...")
1440 WAIT 3
1450 !
1460 ! Reset the cable loss and velocity factor
1470 OUTPUT @Hp87xx;"SENS1:CORR:LOSS:COAX .0"
1480 OUTPUT @Hp87xx;"SENS1:CORR:RVEL:COAX 1."
1490 !
1500 Clear_disp
1510 Disp_mess("Setting units to Meters")
1520 !
1530 ! Set the units to read in METERS
1540 OUTPUT @Hp87xx;"SENS:DIST:UNIT MET"
1550 !OUTPUT @Hp87xx;"SENS:DIST:UNIT FEET"  ! Display units in feet
1560 WAIT 5
1570 !
1580 !
1590 Clear_disp
1600 Disp_mess("Setting Start and Stop Distance")
1610 !
1620 ! Set the start distance to 0.
1630 OUTPUT @Hp87xx;"SENS1:DIST:STAR 0; *WAI"
1640 !
1650 ! Set the stop distance to 100.
1660 OUTPUT @Hp87xx;"SENS1:DIST:STOP 100; *WAI"
1670 !
1680 ! Send an operation complete query to ensure that
1690 ! all overlapped commands have been executed.
1700 OUTPUT @Hp87xx;"*OPC?"
1710 !
1720 ! Wait for the reply.
1730 ENTER @Hp87xx;Opc
1740 !
1750 WAIT 10
1760 !
```

```
1770 ! Change to Band pass mode
1780 OUTPUT @Hp87xx;"SENS:FREQ:MODE CENT; *WAI"
1790 !
1800 Clear_disp
1810 Disp_mess("Setting CF to 600 MHz. Band Pass")
1820 !
1830 ! Set Center Frequency to 600 MHz
1840 OUTPUT @Hp87xx;"SENS1:FREQ:CENT 600000000 HZ;*WAI"
1850 WAIT 10
1860 !
1870 Clear_disp
1880 Disp_mess("Setting CF to 900 MHz. Band Pass")
1890 !
1900 ! Set Center Frequency to 900 MHz
1910 OUTPUT @Hp87xx;"SENS1:FREQ:CENT 900000000 HZ;*WAI"
1920 WAIT 10
1930 !
1940 Clear_disp
1950 Disp_mess("Return to Low Pass Mode")
1960 !
1970 ! Return to Low Pass Mode
1980 OUTPUT @Hp87xx;"SENS:FREQ:MODE LOWP; *WAI"
1990 WAIT 10
2000 !
2010 Clear_disp
2020 Disp_mess("Set Cable Loss to 10dB/100 ft")
2030 OUTPUT @Hp87xx;"SENS1:CORR:LOSS:COAX 10.0"
2040 WAIT 10
2050 !
2060 Clear_disp
2070 Disp_mess("Set Velocity factor to .8")
2080 OUTPUT @Hp87xx;"SENS1:CORR:RVEL:COAX .8"
2090 WAIT 10
2100 !
2110 Clear_disp
2120 Disp_mess("Set   Cable Loss=0., VF=1.0")
2130 OUTPUT @Hp87xx;"SENS1:CORR:LOSS:COAX .0"
2140 OUTPUT @Hp87xx;"SENS1:CORR:RVEL:COAX 1."
2150 WAIT 10
2160 !
2170 DISP "Done"
```

```
2180 BEEP
2190 END
2200 !
2210 SUB Disp_mess(Message$)
2220     COM /Sys_state/ @Hp87xx,Scode
2230     OUTPUT @Hp87xx;"DISP:ANN:MESS:DATA '"&Message$&"'"
2240 SUBEND
2250 !
2260 SUB Clear_disp
2270     COM /Sys_state/ @Hp87xx,Scode
2280     DIM Command$[40]
2290     OUTPUT @Hp87xx;"DISP:ANN:MESS:CLE"
2300 SUBEND
2310 !
2320 !************************************************************
2330 ! Iden_port:   Identify io port to use.
2340 ! Description: This routines sets up the I/O port address for
2350 !              the SCPI interface.  For "HP 87xx" instruments,
2360 !              the address assigned to @Hp87xx = 800 otherwise,
2370 !              716.
2380 !************************************************************
2390 SUB Iden_port
2400     COM /Sys_state/ @Hp87xx,Scode
2410 !
2420     IF POS(SYSTEM$("SYSTEM ID"),"HP 87")<>0 THEN
2430         ASSIGN @Hp87xx TO 800
2440         Scode=8
2450     ELSE
2460         ASSIGN @Hp87xx TO 716
2470         Scode=7
2480     END IF
2490 !
2500 SUBEND !Iden_port
2510 !
```

USR_FLOC Example Program

```
10   ! ----------------------------------------------------------
20   !
30   ! BASIC program: USR_FLOC
40   !
50   ! Fault Location measurements require option 100.
60   ! User BEGIN requires option 1C2, IBASIC.
70   !
80   !
90   ! This is an example user BEGIN program for fault location.
100  !
110  ! Load this program into the analyzer.  Then press [BEGIN]
120  ! [User BEGIN ON].
130  !
140  ! The following line is required. DO NOT REMOVE!
150 User_begin:ASSIGN @Rfna TO 800       ![User Begin] Program
160   ASSIGN @Hp8712 TO 800
170  !
180  ! To Modify:
190  ! Use [IBASIC][EDIT] or [IBASIC][Key Record]
200  !
210  !
220  ! Declare storage for variables.
230   DIM Name$[60],Str1$[60],Str2$[60],Str3$[60]
240  !
250  ! Clear the softkey labels
260   OUTPUT @Rfna;"DISP:MENU2:KEY8 '';*WAI"
270  !
280  ! Re-define softkey labels here.
290   OUTPUT @Rfna;"DISP:MENU2:KEY1 'Test End  of Cable';*WAI"
300   OUTPUT @Rfna;"DISP:MENU2:KEY2 '*';*WAI"
310   OUTPUT @Rfna;"DISP:MENU2:KEY3 'Mkr -> Max';*WAI"
320   OUTPUT @Rfna;"DISP:MENU2:KEY4 'Next Peak Left';*WAI"
330   OUTPUT @Rfna;"DISP:MENU2:KEY5 'Next Peak Right';*WAI"
340   OUTPUT @Rfna;"DISP:MENU2:KEY6 'Zoom on   Marker';*WAI"
350   OUTPUT @Rfna;"DISP:MENU2:KEY7 '*';*WAI"
360  !
```

```
370  !The following 2 lines are required. DO NOT REMOVE!
380 User_pause:PAUSE
390   GOTO User_pause
400  !
410 User_key1:        ! Example Set Stop Distance to 1100ft
420   OUTPUT @Hp8712;"SENS1:STAT ON; *WAI"
430   OUTPUT @Hp8712;"SENS1:FUNC 'FLOC 1,0';DET NBAN; *WAI"
440   OUTPUT @Hp8712;"SENS1:DIST:STOP 1100; *opc?"
450   ENTER @Hp8712;Opc
460   OUTPUT @Hp8712;"SENS1:CORR:RVEL:COAX  0.89"
470   OUTPUT @Hp8712;"DISP:WIND1:TRAC:Y:AUTO ONCE"
480   GOTO User_pause
490  !
500 User_key2:        ! Define softkey 2 here.
510   GOSUB Message ! Remove this line
520   GOTO User_pause
530  !
540 User_key3:        ! Example Marker Function
550   OUTPUT @Rfna;"CALC1:MARK1 ON"
560   OUTPUT @Rfna;"CALC1:MARK:FUNC MAX"
570   GOTO User_pause
580  !
590 User_key4:        ! Define softkey 6 here.
600   OUTPUT @Rfna;"CALC1:MARK1 ON"
610   OUTPUT @Hp8712;"CALC1:MARK:MAX:LEFT"
620   GOTO User_pause
630  !
640 User_key5:        ! Define softkey 5 here.
650   OUTPUT @Hp8712;"CALC1:MARK1 ON"
660   OUTPUT @Hp8712;"CALC1:MARK:MAX:RIGHT"
670   GOTO User_pause
680  !
690 User_key6:        ! Zoom on Cable
700   OUTPUT @Hp8712;"SENS1:STAT ON; *WAI"
710   OUTPUT @Hp8712;"SENS1:FUNC 'FLOC 1,0';DET NBAN; *WAI"
720   OUTPUT @Hp8712;"calc1:mark1:x?"
730   ENTER @Hp8712;Distance
740   New_start=Distance-20
750   IF (New_start<0) THEN New_start=0
760   OUTPUT @Hp8712;"sens1:dist:start "&VAL$(New_start)
770   OUTPUT @Hp8712;"sens1:dist:stop "&VAL$(Distance+20)
```

```
780   OUTPUT @Hp8712;"*opc?"
790   ENTER @Hp8712;Opc
800   GOTO User_pause
810  !
820 User_key7:        ! Define softkey 7 here.
830   GOSUB Message ! Remove this line.
840   GOTO User_pause
850  !
860 Message:     !
870   Str1$="This key is programmable."
880   Str2$="To modify, select"
890   Str3$="[System Options], [IBASIC], [Edit]."
900   OUTPUT @Rfna;"DISP:ANN:MESS
      '"&Str1$&CHR$(10)&Str2$&CHR$(10)&Str3$&"', MEDIUM"
910   RETURN
920  !
930   END
```

Multiport Test Set Measurements

PORT_SEL	Using graphics to show internal connections of the HP 87075C when different ports are selected.
TSET_CAL	Recalling "TSET_CAL.CAL" and performing a test set calibration.

PORT_SELection Example Program

This program displays the internal connections of the HP 87075C multiport test set when different ports are selected. The internal connections of the multiport test set are drawn on the IBASIC display. Whenever the user selects a different port on the multiport test set, the program will redraw the internal connections.

This program also demonstrates how to use IBASIC to draw a fairly complicated drawing on the network analyzer.

> **NOTE**
>
> This program only works with the HP 87075C multiport test set and IBASIC, Option 1C2.

```
1000 ! Filename:  PORT_SEL, 87075 Port Selection Example
1010 !
1020 ! Description:
1030 !      This program demonstrate how the internal connections of 87075
1040 !      are carried out when the different ports for Reflection and
1050 !      Transmission are selected.  It intends to show as an example
1060 !      of how to select the 87075 ports and how to draw draw graphics
1070 !      on the IBASIC window.
1080 !
1090 !
1100 ! NOTE:  This program works properly ONLY
1110 ! when option 1C2, IBASIC, has been installed.
1120 ! Modify to use DISP:WIND";VAL$(Wind);" if no IBASIC option.
1130 !
1140 !
1150
!------------------------------------------------------------------
1160 ! Common Variables
1170 COM /Sys_state/ @Hp87xx,Scode,Keycode,Wind
```

```
1180 COM /Hp8711_coord/
     Orig_x_8711,Orig_y_8711,Refl_x_8711,Refl_y_8711,Tran_x_8711,
     Tran_y_8711
1190 COM /Hp87075_coord/
     Orig_x_87075,Orig_y_87075,Refl_x_87075,Refl_y_87075,Tran_x_87075,
     Tran_y_87075
1200 COM /Hp87075_ports/ Port_x(1:12),Port_y(1:12)
1210 COM /Color/ Erase,Bright,Dim
1220 COM /Sys_var/ Refl_port,Tran_port
1230
!----------------------------------------------------------------------
1240 ! Identify I/O Port
1250 CALL Iden_port
1260 !
1270 OUTPUT @Hp87xx;"SYST:PRES; *WAI"       ! Preset the system
1280 CALL Setup_constant
1290 !
1300 ! Allocate an IBASIC display partition to show the graphics
1310 !
1320 OUTPUT @Hp87xx;"CONT1:MULT:STATE ON"   ! Make sure 87075 mode is
     enabled
1330 OUTPUT @Hp87xx;"DISP:FORM SING"
1340 OUTPUT @Hp87xx;"DISP:PROG:MODE FULL"
1350 OUTPUT @Hp87xx;"DISP:WIND";VAL$(Wind);":GRAP:SCAL 0,1023,0,383"
1360 !
1370 ! Clear the IBASIC display partition.
1380 OUTPUT @Hp87xx;"DISP:WIND";VAL$(Wind);":GRAP:CLE"
1390 !
1400 CALL Draw_analyzer
1410 CALL Draw_87075
1420 !
1430 ! Connect HP8711 to HP87075 for the Refl and Tran ports
1440 CALL Connect(Refl_x_8711,Refl_y_8711,Refl_x_87075,Refl_y_87075,Bright)
1450 CALL Connect(Tran_x_8711,Tran_y_8711,Tran_x_87075,Tran_y_87075,Bright)
1460 CALL Set_refl(1)
1470 CALL Set_tran(2)
1480 !
1490 ! Infinite loop to wait for softkey requests
1500 Do_loop:!
1510 GOSUB Setup_srq
1520 !
```

```
1530 GOTO Do_loop
1540 STOP
1550 !
1560 !----------------------------------------------------------------------
1570 ! Setup interrupts
1580 !
1590 Setup_srq:!
1600 ! If using an external controller...
1610 !
1620 ! Initialize flag for checking on keyboard
1630 ! interrupts.
1640 Keycode=-1
1650 !
1660 ! Label softkey 1.
1670 OUTPUT @Hp87xx;"DISP:MENU:KEY1 'Reflection to Port #'"
1680 OUTPUT @Hp87xx;"DISP:MENU:KEY2 'Transmissn to Port #'"
1690 OUTPUT @Hp87xx;"DISP:MENU:KEY5 'Done'"
1700 !
1710 ! Clear the status register and event status
1720 ! register.
1730 OUTPUT @Hp87xx;"*CLS;*ESE 0"
1740 ! Preset the other status registers.
1750 ! Enable the Device Status register to report
1760 ! to the Status Byte on positive transition
1770 ! of bit 0 (key press).  Enable the Status
1780 ! Byte to generate an interrupt when the
1790 ! Device Status register's summary bit
1800 ! changes.
1810 OUTPUT @Hp87xx;"STAT:PRES;DEV:ENAB 1;*SRE 4"
1820 !
1830 ! Clear the key queue to ensure that previous
1840 ! key presses  do not generate an interrupt.
1850 OUTPUT @Hp87xx;"SYST:KEY:QUE:CLE"
1860 !
1870 ! Set up and enable the interrupt on the HP-IB
1880 ! when a service request is received.
1890 ON INTR Scode,5 RECOVER Srq
1900 ENABLE INTR Scode;2
1910 Suspend: !WAIT 5                      ! Use WAIT 'n' to suspend IBASIC
1920 GOTO Suspend
1930 !
```

```
1940 !-----------------------------------------------------------------
1950 ! Interrupt Handler
1960 !
1970 Srq:   !
1980 !
1990 ! Do a serial poll to find out if analyzer generated the
2000 ! interrupt.
2010 Stb=SPOLL(@Hp87xx)
2020 !
2030 ! Determine if the Device Status register's summary
2040 ! bit (bit 2 of the Status Byte) has been set.
2050 IF BINAND(Stb,4)<>0 THEN
2060 !
2070 ! If so, then get the Device Status Register contents.
2080     OUTPUT @Hp87xx;"STAT:DEV:EVEN?"
2090     ENTER @Hp87xx;Dev_event
2100 !
2110 ! Check for key press...
2120     IF BINAND(Dev_event,1)<>0 THEN
2130 ! If so, then determine which key.
2140         OUTPUT @Hp87xx;"SYST:KEY?"
2150         ENTER @Hp87xx;Keycode
2160     END IF
2170 END IF
2180 !
2190 ! Reenable the interrupt in case wrong key
2200 ! was pressed.
2210 CALL Softkey_handler
2220 ENABLE INTR Scode
2230 !
2240 RETURN
2250 END
2260 !
2270 !-----------------------------------------------------------------
2280 ! Subroutines
2290 !
2300 !
2310 ! Setup_constant
2320 !   Setup all global constants
2330 SUB Setup_constant
2340     COM /Sys_state/ @Hp87xx,Scode,Keycode,Wind
```

```
2350     COM /Hp8711_coord/
         Orig_x_8711,Orig_y_8711,Refl_x_8711,Refl_y_8711,Tran_x_8711,
         Tran_y_8711
2360     COM /Hp87075_coord/
         Orig_x_87075,Orig_y_87075,Refl_x_87075,Refl_y_87075,Tran_x_87075,
         Tran_y_87075
2370     COM /Hp87075_ports/ Port_x(1:12),Port_y(1:12)
2380     COM /Color/ Erase,Bright,Dim
2390     COM /Sys_var/ Refl_port,Tran_port
2400     Orig_x_8711=30
2410     Orig_y_8711=170
2420     Refl_x_8711=Orig_x_8711+300
2430     Refl_y_8711=Orig_y_8711+20
2440     Tran_x_8711=Orig_x_8711+410
2450     Tran_y_8711=Orig_y_8711+20
2460     Orig_x_87075=30
2470     Orig_y_87075=30
2480     Refl_x_87075=Orig_x_87075+300
2490     Refl_y_87075=Orig_y_87075+80
2500     Tran_x_87075=Orig_x_87075+410
2510     Tran_y_87075=Orig_y_87075+80
2520     FOR I=1 TO 11 STEP 2
2530         Port_x(I)=Orig_x_87075+100+(((I-1)/2)*60)
2540         Port_y(I)=Orig_y_87075+45
2550     NEXT I
2560     FOR I=2 TO 12 STEP 2
2570         Port_x(I)=Orig_x_87075+130+(((I-2)/2)*60)
2580         Port_y(I)=Orig_y_87075+25
2590     NEXT I
2600     Erase=0
2610     Bright=1
2620     Dim=2
2630     Wind=10
2640     Refl_port=0
2650     Tran_port=0
2660 SUBEND
2670 !
2680 !-------------------------------------------------------------------
2690 !  Drawing routines
2700 !
2710 !  Draw_analyzer
```

```
2720 !   Draw an HP8711 Analyzer on the Ibasic window
2730 SUB Draw_analyzer
2740     COM /Sys_state/ @Hp87xx,Scode,Keycode,Wind
2750     COM /Hp8711_coord/
         Orig_x_8711,Orig_y_8711,Refl_x_8711,Refl_y_8711,Tran_x_8711,
         Tran_y_8711
2760 ! Select the bright "pen" and bold font.
2770     OUTPUT @Hp87xx;"DISP:WIND";VAL$(Wind);":GRAP:COL 1;LAB:FONT BOLD"
2780 !
2790 ! Draw a label reading "HP 8711" at 30 pixels
2800 ! to the right and 270 pixels above the origin.
2810 ! The origin is the lower left corner of the
2820 ! current graphics window
2830     OUTPUT @Hp87xx;"DISP:WIND";VAL$(Wind);":GRAP:MOVE
         ";Orig_x_8711;",";Orig_y_8711+120+10;";LAB 'HP 8711C'"
2840 !
2850 ! Draw a box to represent the analyzer.
2860     OUTPUT @Hp87xx;"DISP:WIND";VAL$(Wind);":GRAP:MOVE
         ";Orig_x_8711;",";Orig_y_8711
2870     OUTPUT @Hp87xx;"DISP:WIND";VAL$(Wind);":GRAP:RECT 480,120"
2880     OUTPUT @Hp87xx;"DISP:WIND";VAL$(Wind);":GRAP:MOVE
         ";Orig_x_8711+20;",";Orig_y_8711+10
2890     OUTPUT @Hp87xx;"DISP:WIND";VAL$(Wind);":GRAP:RECT 210,100"
2900     OUTPUT @Hp87xx;"DISP:WIND";VAL$(Wind);":GRAP:MOVE
         ";Refl_x_8711;",";Refl_y_8711
2910     OUTPUT @Hp87xx;"DISP:WIND";VAL$(Wind);":GRAP:CIRC 4"
2920     OUTPUT @Hp87xx;"DISP:WIND";VAL$(Wind);":GRAP:MOVE
         ";Tran_x_8711;",";Tran_y_8711
2930     OUTPUT @Hp87xx;"DISP:WIND";VAL$(Wind);":GRAP:CIRC 4"
2940     OUTPUT @Hp87xx;"DISP:WIND";VAL$(Wind);":GRAP:COL 1;LAB:FONT SLAN"
2950     OUTPUT @Hp87xx;"DISP:WIND";VAL$(Wind);":GRAP:MOVE
         ";Refl_x_8711-40;",";Refl_y_8711+10
2960     OUTPUT @Hp87xx;"DISP:WIND";VAL$(Wind);":GRAP:LAB 'RF OUT'"
2970     OUTPUT @Hp87xx;"DISP:WIND";VAL$(Wind);":GRAP:COL 1;LAB:FONT SLAN"
2980     OUTPUT @Hp87xx;"DISP:WIND";VAL$(Wind);":GRAP:MOVE
         ";Tran_x_8711-20;",";Tran_y_8711+10
2990     OUTPUT @Hp87xx;"DISP:WIND";VAL$(Wind);":GRAP:LAB 'RF IN'"
3000 SUBEND
3010 !
3020 ! Draw_87075
3030 !   Draw an 87075 Multiport test set with twelve port setups
```

```
3040 SUB Draw_87075
3050     COM /Sys_state/ @Hp87xx,Scode,Keycode,Wind
3060     COM /Hp87075_coord/
         Orig_x_87075,Orig_y_87075,Refl_x_87075,Refl_y_87075,Tran_x_87075,
         Tran_y_87075
3070     COM /Hp87075_ports/ Port_x(1:12),Port_y(1:12)
3080 ! Select the bright "pen" and bold font.
3090     OUTPUT @Hp87xx;"DISP:WIND";VAL$(Wind);":GRAP:COL 1;LAB:FONT BOLD"
3100 !
3110 !
3120 ! Draw a label reading "HP 87075" at 30 pixels
3130 ! to the right and 110 pixels above the origin.
3140 ! The origin is the lower left corner of the
3150 ! current graphics window
3160     OUTPUT @Hp87xx;"DISP:WIND";VAL$(Wind);":GRAP:MOVE
         ";Orig_x_87075;",";Orig_y_87075+100+10;";LAB 'HP 87075'"
3170 !
3180     OUTPUT @Hp87xx;"DISP:WIND";VAL$(Wind);":GRAP:MOVE
         ";Orig_x_87075;",";Orig_y_87075
3190     OUTPUT @Hp87xx;"DISP:WIND";VAL$(Wind);":GRAP:RECT 480,110"
3200     OUTPUT @Hp87xx;"DISP:WIND";VAL$(Wind);":GRAP:MOVE
         ";Refl_x_87075;",";Refl_y_87075
3210     OUTPUT @Hp87xx;"DISP:WIND";VAL$(Wind);":GRAP:CIRC 4"
3220     OUTPUT @Hp87xx;"DISP:WIND";VAL$(Wind);":GRAP:MOVE
         ";Tran_x_87075;",";Tran_y_87075
3230     OUTPUT @Hp87xx;"DISP:WIND";VAL$(Wind);":GRAP:CIRC 4"
3240     OUTPUT @Hp87xx;"DISP:WIND";VAL$(Wind);":GRAP:COL 1;LAB:FONT SLAN"
3250     OUTPUT @Hp87xx;"DISP:WIND";VAL$(Wind);":GRAP:MOVE
         ";Refl_x_87075-40;",";Refl_y_87075+10
3260     OUTPUT @Hp87xx;"DISP:WIND";VAL$(Wind);":GRAP:LAB 'REFL'"
3270     OUTPUT @Hp87xx;"DISP:WIND";VAL$(Wind);":GRAP:COL 1;LAB:FONT SLAN"
3280     OUTPUT @Hp87xx;"DISP:WIND";VAL$(Wind);":GRAP:MOVE
         ";Tran_x_87075-20;",";Tran_y_87075+10
3290     OUTPUT @Hp87xx;"DISP:WIND";VAL$(Wind);":GRAP:LAB 'TRAN'"
3300     FOR I=1 TO 12
3310         OUTPUT @Hp87xx;"DISP:WIND";VAL$(Wind);":GRAP:MOVE
             ";Port_x(I);",";Port_y(I)
3320         OUTPUT @Hp87xx;"DISP:WIND";VAL$(Wind);":GRAP:CIRC 4"
3330         OUTPUT @Hp87xx;"DISP:WIND";VAL$(Wind);":GRAP:COL 1;LAB:FONT
             SLAN"
3340         OUTPUT @Hp87xx;"DISP:WIND";VAL$(Wind);":GRAP:MOVE
```

```
        ";Port_x(I)-8;",";Port_y(I)-18
3350        OUTPUT @Hp87xx;"DISP:WIND";VAL$(Wind);":GRAP:LAB '";VAL$(I);"'"
3360      NEXT I
3370 SUBEND
3380 !
3390 !-----------------------------------------------------------------
3400 ! Connection routines
3410 !
3420 ! Connect
3430 !   Connect (x1,y1) to (x2,y2) with the specied color 'Col'
3440 !   If Color = 0, it will be an erase command instead.
3450 SUB Connect(X1,Y1,X2,Y2,Col)
3460     COM /Sys_state/ @Hp87xx,Scode,Keycode,Wind
3470     OUTPUT @Hp87xx;"DISP:WIND";VAL$(Wind);":GRAP:COL ";Col
3480     OUTPUT @Hp87xx;"DISP:WIND";VAL$(Wind);":GRAP:MOVE ";X1;",";Y1
3490     OUTPUT @Hp87xx;"DISP:WIND";VAL$(Wind);":GRAP:DRAW ";X2;",";Y2
3500 SUBEND
3510 !
3520 ! Connect_refl
3530 !   Connect the reflection port to the specified port index I with
3540 !   color 'Col'.  Use Col=0 to erase the connection.  This routine
3550 !   uses the port coordinates from Port_x(1:12) and Port_y(1:12)
3560 SUB Connect_refl(I,Col)
3570     COM /Sys_state/ @Hp87xx,Scode,Keycode,Wind
3580     COM /Hp8711_coord/
         Orig_x_8711,Orig_y_8711,Refl_x_8711,Refl_y_8711,Tran_x_8711,
         Tran_y_8711
3590     COM /Hp87075_coord/
         Orig_x_87075,Orig_y_87075,Refl_x_87075,Refl_y_87075,Tran_x_87075,
         Tran_y_87075
3600     COM /Hp87075_ports/ Port_x(1:12),Port_y(1:12)
3610 !
3620     Temp_y=Refl_y_87075-10
3630     Connect(Refl_x_87075,Refl_y_87075,Refl_x_87075,Temp_y,Col)
3640     Connect(Refl_x_87075,Temp_y,Port_x(I),Temp_y,Col)
3650     Connect(Port_x(I),Temp_y,Port_x(I),Port_y(I),Col)
3660 SUBEND
3670 !
3680 !
3690 ! Connect_tran
3700 !   Connect the transmission port to the specified port index I with
```

```
3710 !   color 'Col'.  Use Col=0 to erase the connection.  This routine
3720 !   uses the port coordinates from Port_x(1:12) and Port_y(1:12)
3730 SUB Connect_tran(I,Col)
3740     COM /Sys_state/ @Hp87xx,Scode,Keycode,Wind
3750     COM /Hp8711_coord/
         Orig_x_8711,Orig_y_8711,Refl_x_8711,Refl_y_8711,Tran_x_8711,
         Tran_y_8711
3760     COM /Hp87075_coord/
         Orig_x_87075,Orig_y_87075,Refl_x_87075,Refl_y_87075,Tran_x_87075,
         Tran_y_87075
3770     COM /Hp87075_ports/ Port_x(1:12),Port_y(1:12)
3780 !
3790     Temp_y=Refl_y_87075-20
3800     Connect(Tran_x_87075,Tran_y_87075,Tran_x_87075,Temp_y,Col)
3810     Connect(Tran_x_87075,Temp_y,Port_x(I),Temp_y,Col)
3820     Connect(Port_x(I),Temp_y,Port_x(I),Port_y(I),Col)
3830 SUBEND
3840 !
3850 !-----------------------------------------------------------------
3860 ! Softkey handle routines
3870 !
3880 ! Select_refl
3890 !   Select the reflection port by requesting a valid port number from
3900 !   the user.  The input port number is used to select the reflection
3910 !   port accordingly.  This routine will also update the drawing
3920 !   connections on the Ibasic window.  Any invalid number will be
3930 !   ignored.
3940 SUB Select_refl
3950     COM /Sys_state/ @Hp87xx,Scode,Keycode,Wind
3960     COM /Hp8711_coord/
         Orig_x_8711,Orig_y_8711,Refl_x_8711,Refl_y_8711,Tran_x_8711,
         Tran_y_8711
3970     COM /Hp87075_coord/
         Orig_x_87075,Orig_y_87075,Refl_x_87075,Refl_y_87075,Tran_x_87075,
         Tran_y_87075
3980     COM /Color/ Erase,Bright,Dim
3990     COM /Sys_var/ Refl_port,Tran_port
4000 !
4010     OUTPUT @Hp87xx;"DISP:WIND";VAL$(Wind);":GRAP:COL ";Bright
4020     OUTPUT @Hp87xx;"*OPC?"
4030     ENTER @Hp87xx;Opc
```

```
4040     INPUT "Connect Reflection to Port #:",P
4050     CALL Set_refl(P)
4060 SUBEND
4070 !
4080 !
4090 ! Set_refl
4100 !   Update the currently selected reflection port with the specified
4110 !   port 'P'.  Update the connection drawing on the Ibasic window.
4120 SUB Set_refl(P)
4130     COM /Sys_state/ @Hp87xx,Scode,Keycode,Wind
4140     COM /Hp8711_coord/
         Orig_x_8711,Orig_y_8711,Refl_x_8711,Refl_y_8711,Tran_x_8711,
         Tran_y_8711
4150     COM /Hp87075_coord/
         Orig_x_87075,Orig_y_87075,Refl_x_87075,Refl_y_87075,Tran_x_87075,
         Tran_y_87075
4160     COM /Color/ Erase,Bright,Dim
4170     COM /Sys_var/ Refl_port,Tran_port
4180 !
4190     OUTPUT @Hp87xx;"ROUT:REFL:PATH:DEFine:PORT ";P
4200     OUTPUT @Hp87xx;"ROUT:REFL:PATH:DEFine:PORT?"
4210     ENTER @Hp87xx;New_refl
4220     OUTPUT @Hp87xx;"ROUT:TRAN:PATH:DEFine:PORT?"
4230     ENTER @Hp87xx;New_tran
4240     Update_ports(New_refl,New_tran)
4250 SUBEND
4260 !
4270 !
4280 !
4290 ! Select_tran
4300 !   Select the transmission port by requesting a valid port number from
4310 !   the user.  The input port number is used to select the transmission
4320 !   port accordingly.  This routine will also update the drawing
4330 !   connections on the Ibasic window.  Any invalid number will be
4340 !   ignored.
4350 SUB Select_tran
4360     COM /Sys_state/ @Hp87xx,Scode,Keycode,Wind
4370     COM /Hp8711_coord/
         Orig_x_8711,Orig_y_8711,Refl_x_8711,Refl_y_8711,Tran_x_8711,
         Tran_y_8711
4380     COM /Hp87075_coord/
```

```
        Orig_x_87075,Orig_y_87075,Refl_x_87075,Refl_y_87075,Tran_x_87075,
        Tran_y_87075
4390    COM /Color/ Erase,Bright,Dim
4400    COM /Sys_var/ Refl_port,Tran_port
4410 !
4420    OUTPUT @Hp87xx;"DISP:WIND";VAL$(Wind);":GRAP:COL ";Bright
4430    OUTPUT @Hp87xx;"*OPC?"
4440    ENTER @Hp87xx;Opc
4450    INPUT "Connect Transmission to Port #:",P
4460    CALL Set_tran(P)
4470 SUBEND
4480 !
4490 !
4500 ! Set_tran
4510 !   Update the currently selected transmission port with the specified
4520 !   port 'P'.  Update the connection drawing on the Ibasic window.
4530 SUB Set_tran(P)
4540    COM /Sys_state/ @Hp87xx,Scode,Keycode,Wind
4550    COM /Hp8711_coord/
        Orig_x_8711,Orig_y_8711,Refl_x_8711,Refl_y_8711,Tran_x_8711,
        Tran_y_8711
4560    COM /Hp87075_coord/
        Orig_x_87075,Orig_y_87075,Refl_x_87075,Refl_y_87075,Tran_x_87075,
        Tran_y_87075
4570    COM /Color/ Erase,Bright,Dim
4580    COM /Sys_var/ Refl_port,Tran_port
4590 !
4600    OUTPUT @Hp87xx;"ROUT:TRAN:PATH:DEFine:PORT ";P
4610    OUTPUT @Hp87xx;"ROUT:TRAN:PATH:DEFine:PORT?"
4620    ENTER @Hp87xx;New_tran
4630    OUTPUT @Hp87xx;"ROUT:REFL:PATH:DEFine:PORT?"
4640    ENTER @Hp87xx;New_refl
4650    Update_ports(New_refl,New_tran)
4660 SUBEND
4670 !
4680 ! Update_ports
4690 !   Update the currently selected ports.  Erase old connections.
4700 !   Draw new connections.
4710 SUB Update_ports(Refl,Tran)
4720    COM /Color/ Erase,Bright,Dim
4730    COM /Sys_var/ Refl_port,Tran_port
```

```
4740     IF Tran_port=0 THEN
4750         Tran_port=Tran
4760     ELSE
4770         IF Tran<>Tran_port THEN
4780             Connect_tran(Tran_port,Erase)
4790             Tran_port=Tran
4800         END IF
4810     END IF
4820     IF Refl_port=0 THEN
4830         Refl_port=Refl
4840     ELSE
4850         IF Refl<>Refl_port THEN
4860             Connect_refl(Refl_port,Erase)
4870             Refl_port=Refl
4880         END IF
4890     END IF
4900     Connect_tran(Tran_port,Dim)
4910     Connect_refl(Refl_port,Dim)
4920 SUBEND
4930 !
4940 ! Softkey_handler
4950 !   Call from Srq to handler all softkey requests.  Terminate program
4960 !   when 'Done' is pressed.
4970 SUB Softkey_handler
4980     COM /Sys_state/ @Hp87xx,Scode,Keycode,Wind
4990 !
5000     IF Keycode=0 THEN
5010         CALL Select_refl
5020     ELSE
5030         IF Keycode=1 THEN
5040             CALL Select_tran
5050         ELSE
5060             IF Keycode=4 THEN
5070                 OUTPUT @Hp87xx;"SYST:PRES; *WAI"       ! Preset the
                     system^M
5080                 STOP
5090             END IF
5100         END IF
5110     END IF
5120 SUBEND
5130 !
```

```
5140 !-----------------------------------------------------------------
5150 ! Misc routines
5160 !
5170 !*********************************************************
5180 ! Iden_port:   Identify io port to use.
5190 ! Description: This routines sets up the I/O port address for
5200 !              the SCPI interface.  For "HP 87xx" instruments,
5210 !              the address assigned to @Hp87xx = 800 otherwise,
5220 !              716.
5230 !*********************************************************
5240 SUB Iden_port
5250     COM /Sys_state/ @Hp87xx,Scode,Keycode,Wind
5260 !
5270     IF POS(SYSTEM$("SYSTEM ID"),"HP 87")<>0 THEN
5280         ASSIGN @Hp87xx TO 800
5290         Scode=8
5300     ELSE
5310         ASSIGN @Hp87xx TO 716
5320         Scode=7
5330     END IF
5340 !
5350 SUBEND!Iden_port
5360 !
```

TSET_CAL Example Program

This program automates the process of recalling and performing an HP 87075C multiport test set cal. This program first attempts to recall "TSET_CAL.CAL" from non-volatile RAM, then the internal disk drive. If the recall is successful, it invokes the recalled test set cal for transmission and reflection of measurement channels 1 and 2.

> **NOTE**
>
> This program only works with the HP 87075C multiport test set.

```
1000 ! Filename: TSET_CAL, recall test set cal
1010 !
1020 ! Description:
1030 !    This program will try to recall tset_cal.cal from NVRAM if file
1040 !    is present.  If not, it will then try to recall tset_cal.cal from
1050 !    INT device instead.  If recall successful, it will then do test
1060 !    set cal for Transmission and Reflection of Channel 1 and Channel
          2.
1070
!-----------------------------------------------------------------------
1080 ! Common Variables
1090 COM /Sys_state/ @Hp87xx,Scode,Errnum
1100 ! Identify I/O Port
1110 CALL Iden_port
1120 OUTPUT @Hp87xx;"syst:pres;*wai"      ! Reset the instrument
1130 CALL Recall_tset_cal
1140 IF (Errnum=0) THEN
1150     PRINT "Doing Test set cal................"
1160     CALL Tsetcal
1170     PRINT "Test Set cal complete"
1180 END IF
```

```
1190 !
1200 OUTPUT @Hp87xx;"syst:pres;*wai"      ! Reset the instrument
1210 !
1220 STOP
1230 END
1240 !
1250 !***************************************************************
1260 ! TsetCal:   Do Test set Cal.
1270 ! Description:  This routine will do test set cal for both
1280 !               Transmission and Reflection of Channel 1 and
1290 !               Channel 2.
1300 !***************************************************************
1310 SUB Tsetcal
1320    COM /Sys_state/ @Hp87xx,Scode,Errnum
1330 !
1340 !   Do Test set cal for channel 1 Transmission
1350    OUTPUT @Hp87xx;"sens1:stat ON; *wai"
1360    OUTPUT @Hp87xx;"sens1:func 'xfr:pow:rat 2,0';det nban; *wai"
1370    OUTPUT @Hp87xx;"sens1:corr:testset; *wai"
1380 !
1390 !   Do Test Set cal for channel 1 Reflection
1400    OUTPUT @Hp87xx;"sens1:stat ON; *wai"
1410    OUTPUT @Hp87xx;"sens1:func 'xfr:pow:rat 1,0';det nban; *wai"
1420    OUTPUT @Hp87xx;"sens1:corr:testset; *wai"
1430 !
1440 !   Do Test set cal for channel 2 Transmission
1450    OUTPUT @Hp87xx;"sens2:stat ON; *wai"
1460    OUTPUT @Hp87xx;"sens2:func 'xfr:pow:rat 2,0';det nban; *wai"
1470    OUTPUT @Hp87xx;"sens2:corr:testset; *wai"
1480 !
1490 !   Do Test Set cal for channel 2 Reflection
1500    OUTPUT @Hp87xx;"sens2:stat ON; *wai"
1510    OUTPUT @Hp87xx;"sens2:func 'xfr:pow:rat 1,0';det nban; *wai"
1520    OUTPUT @Hp87xx;"sens2:corr:testset; *wai"
1530 SUBEND
1540 !
```

```
1550 !*************************************************************
1560 ! Recall_tset_cal:   Recall Test Set Cal.
1570 ! Description:  This routine will try to recall tset_cal.cal
1580 !               from NVRAM if available.  If file not found,
1590 !               it will then try to recall tset_cal.cal from
1600 !               INT device instead.
1610 !*************************************************************
1620 SUB Recall_tset_cal
1630     COM /Sys_state/ @Hp87xx,Scode,Errnum
1640     DIM E$[120]
1650 !
1660 !   Clear the status register and event status
1670 !   register.
1680     OUTPUT @Hp87xx;"*cls;*ese 0"
1690     PRINT "Recalling tset_cal.cal from NVRAM device........."
1700     OUTPUT @Hp87xx;"mmem:load:stat 1,'mem:tset_cal.cal'"
1710     OUTPUT @Hp87xx;"syst:err?"
1720     ENTER @Hp87xx;Errnum,E$
1730     IF (Errnum<>0) THEN
1740         PRINT E$
1750         PRINT "Recalling tset_cal.cal from INT device........."
1760         OUTPUT @Hp87xx;"mmem:load:stat 1,'int:tset_cal.cal'"
1770         OUTPUT @Hp87xx;"syst:err?"
1780         ENTER @Hp87xx;Errnum,E$
1790         IF (Errnum=0) THEN
1800             PRINT "Recall complete"
1810         ELSE
1820             PRINT "TSET_CAL.CAL is not present, recall aborted"
1830         END IF
1840     ELSE
1850         PRINT "Recall complete"
1860     END IF
1870 !
1880 SUBEND!Recall_tset_cal
1890 !
1900 !
1910 !
```

```
1920 !**************************************************************
1930 ! Iden_port:   Identify io port to use.
1940 ! Description: This routines sets up the I/O port address for
1950 !              the SCPI interface.  For "HP 87xx" instruments,
1960 !              the address assigned to @Hp87xx = 800 otherwise,
1970 !              716.
1980 !**************************************************************
1990 SUB Iden_port
2000     COM /Sys_state/ @Hp87xx,Scode,Errnum
2010 !
2020     IF POS(SYSTEM$("SYSTEM ID"),"HP 87")<>0 THEN
2030         ASSIGN @Hp87xx TO 800
2040         Scode=8
2050     ELSE
2060         ASSIGN @Hp87xx TO 716
2070         Scode=7
2080     END IF
2090 !
2100 SUBEND!Iden_port
2110 !
```

TTL Output

TTL_IO Example Program

This program continuously reads the USER TTL port and reports the number of times the port has detected a closure (short) via an external switch. This program is useful in a production environment where a device must be properly connected, either manually or by automated means, where the analyzer must wait for a signal from the operator that the DUT is in place and is ready to be tested.

This program reads the user TTL port continuously until a short (0) is detected. Once this has been detected, a message is displayed. It then waits for the switch to open (1) and displays another message. At this point, code can be added to take a sweep and measure the DUT. The total number of cycles is counted and is displayed.

```
1000 ! Filename: TTL_IO
1010 !
1020 ! This program reads the USER TTL IO
1030 ! port, and counts how many times a
1040 ! switch connected to the port is pressed.
1050 !
1060 DIM Msg$[200]
1070 INTEGER X
1080 !
1090 !
1100 COM /Sys_state/ @Hp87xx,Scode
1110 ! Identify I/O Port
1120 CALL Iden_port
1130 !
1140 !
1150 Pass_count=0
1160 Start: !
1170 LOOP
```

```
1180 ! Display message
1190    Msg$="'DUTs passed: "&VAL$(Pass_count)&CHR$(10)
1200    Msg$=Msg$&"Press button to measure next DUT.'"
1210    OUTPUT @Hp87xx;"DISP:ANN:MESS ";Msg$
1220 !
1230 ! Wait for button to be pressed
1240    REPEAT
1250        OUTPUT @Hp87xx;"DIAG:PORT:READ? 15,1"
1260        ENTER @Hp87xx;X
1270    UNTIL X=0
1280    DISP "Button is now pressed."
1290    OUTPUT @Hp87xx;"DISP:ANN:MESS:CLEAR"
1300 !
1310 ! Wait for button to be released
1320    REPEAT
1330        OUTPUT @Hp87xx;"DIAG:PORT:READ? 15,1"
1340        ENTER @Hp87xx;X
1350    UNTIL X=1
1360    DISP "Button is now released."
1370 !
1380    OUTPUT @Hp87xx;"DISP:ANN:MESS 'Measuring...'"
1390 ! Add code here to take sweep
1400 ! and measure DUT.
1410    WAIT 1
1420    Pass_count=Pass_count+1
1430 END LOOP
1440 END
1450 !
1460 !**************************************************************
1470 ! Iden_port:   Identify io port to use.
1480 ! Description: This routines sets up the I/O port address for
1490 !              the SCPI interface.  For "HP 87xx" instruments,
1500 !              the address assigned to @Hp87xx = 800 otherwise,
1510 !              716.
1520 !**************************************************************
```

```
1530 SUB Iden_port
1540     COM /Sys_state/ @Hp87xx,Scode
1550 !
1560     IF POS(SYSTEM$("SYSTEM ID"),"HP 87")<>0 THEN
1570         ASSIGN @Hp87xx TO 800
1580         Scode=8
1590     ELSE
1600         ASSIGN @Hp87xx TO 716
1610         Scode=7
1620     END IF
1630 !
1640 SUBEND !Iden_port
1650 !
```

AM Delay

AMDELAY Example Program

This program demonstrates the calibration and AM delay measurement of a bandpass filter.

> **NOTE**
>
> This program can only be used with Option 1DA or 1DB.

```
1000 !Filename:  AMDELAY
1010 !
1020 !
1030 ! DESCRIPTION:
1040 !  1) The HP871x analyzer is Preset, CH1 = Y/X, CH2 = AM Delay
1050 !      Pout=10dBm, Fstart=50MHz, Fstop=400MHz, Display is scaled.
1060 !  2) User must connect splitter to RF Out port, connect
1070 !      External detector X to one splitter arm, connect
1080 !      External detector Y to the other splitter arm.
1090 !  3) Analyzer performs an AM Delay calibration over 50-400MHz.
1100 !  4) User must connect the BPF test device between splitter
1110 !      and the External Y detector.
1120 !  5) Analyzer finds the BPF's center frequency and -6dB bandwidth
1130 !      using the Bandwidth marker function.  Fcenter and Fspan
1140 !      are set so the analyzer displays the -6dB bandwidth.
1150 !      (Note:  the analyzer also interpolates numbers for the
1160 !      AM Delay calibration when the frequencies are changed.)
1170 !  6) The Y/X marker 1 is set to max (for maximum transmission.
```

```
1180 !  7) The AM Delay marker 1 is read out and displayed.
1190 !  8) The AM Delay trace is read out, and the first data point
1200 !      is displayed.
1210 !
1220 !*************************************************
1230 ! DEFINITIONS
1240 !
1250 REAL Opc,Freq_center,Freq_span,Q_bpf,Loss_bpf,I,Mrkr_freq
1260 REAL Trace_delay(1:201),Mrkr_delay
1270 !
1280 !*************************************************
1290 ! Determine computer type and assign i/o path
1300 !
1310 CLEAR SCREEN
1320 !
1330 COM /Sys_state/ @Hp87xx,Scode
1340 ! Identify I/O Port
1350 CALL Iden_port
1360 !
1370 !
1380 !-------------------------------------------------
1390 ! Preset analyzer; CH2 = AM Delay; CH1 = External Broadband Y/X; RF
       Power = 10dBm.
1400 !   Note that the Start and Stop frequencies are set before CH2 is set
1410 !   to AM Delay mode.  This is done because when both AM Delay and
1420 !   Y/X are selected, the analyzer switches to alternate sweep mode, so
1430 !   by selecting the frequencies first, the start and stop frequencies
1440 !   will be the same for both channels.
1450 !
1460 OUTPUT @Hp87xx;"SYST:PRES;*OPC?"              !preset instrument
1470 ENTER @Hp87xx;Opc           !waits for PRESET to finish before
     proceeding
1480 DISP "Setting up AM Delay and Y/X measurement..."
1490 !
1500 OUTPUT @Hp87xx;"SENS1:FREQ:STAR 50 MHZ"       !set start freq to
     include test filter
1510 OUTPUT @Hp87xx;"SENS1:FREQ:STOP 400 MHZ"      !set stop freq to
     include test filter
1520 OUTPUT @Hp87xx;"SOUR1:POW 10 DBM"             !set Source Power to
     10dBm
1530 !
```

```
1540 OUTPUT @Hp87xx;"SENS1:FUNC 'XFR:POW:RAT 12,11';DET BBAN;*WAI"
     !CHAN1=broadband Y/X
1550 OUTPUT @Hp87xx;"DISP:WIND1:TRAC:Y:PDIV 1 DB"   !scale Y/X to 1dB/div
1560 OUTPUT @Hp87xx;"DISP:WIND1:TRAC:Y:RLEV -3"     !X/Y ref level =-3dB
1570 !
1580 OUTPUT @Hp87xx;"SENS2:FUNC 'XFR:GDEL:RAT 12,11';DET BBAN;*WAI"
     !CHAN2=AM Delay
1590 OUTPUT @Hp87xx;"SENS2:STAT ON;*WAI"            !turn on CHAN2.  Results
     in Alt Sweep.
1600 OUTPUT @Hp87xx;"SENS2:BWID 250 HZ;*WAI"        !Narrow BW for low noise
1610 OUTPUT @Hp87xx;"DISP:WIND2:TRAC:Y:PDIV 5E-9;*OPC?"   !scale AM Delay to
     5ns/div
1620 ENTER @Hp87xx;Opc                              !wait for commands to
     finish
1630 !
1640 !--------------------------------------------------
1650 ! Calibrate the AM Delay using the "CALIBRATE/Response" function
1660 !
1670 INPUT "CALIBRATE:  Connect Y detector to splitter; press ENTER.",I
1680 DISP "Calibrating..."
1690 OUTPUT @Hp87xx;"SENS2:CORR:COLL:IST OFF;METH TRAN1;*WAI"   !AM Delay
     Response Cal
1700 OUTPUT @Hp87xx;"DISP:ANN:MESS:AOFF"                  !turn off
     message on screen
1710 OUTPUT @Hp87xx;"SENS2:CORR:COLL STAN1;*WAI;:SENS2:CORR:COLL:SAVE;*OPC?"
     !measure standard
1720 ENTER @Hp87xx;Opc                              !wait for calibration to
     finish
1730 INPUT "Insert BPF between splitter and Y detector; press ENTER.",I
1740 !
1750 !----------------------------------------------------
1760 ! Set Center and Span frequencies to -6dB bandwidth of BPF.
1770 !
1780 ! Center the filter's frequency response (to get an accurate Bandwidth
         measurement)
1790 !
1800 DISP "Setting analyzer frequencies..."          !message to user
1810 OUTPUT @Hp87xx;"ABOR;:INIT1:CONT OFF;:INIT1;*OPC?" !perform single
     sweep
1820 ENTER @Hp87xx;Opc                              !wait for sweep to
     finish
```

```
1830 OUTPUT @Hp87xx;"CALC1:MARK:FUNC MAX;*OPC?"     !set Marker to max
1840 ENTER @Hp87xx;Opc                              !wait for marker to move
1850 OUTPUT @Hp87xx;"CALC1:MARK:X?"                 !get Marker frequency
     setting
1860 ENTER @Hp87xx;Mrkr_freq                        !read frequency of max
     marker
1870 OUTPUT @Hp87xx;"SENS1:FREQ:CENT "&VAL$(Mrkr_freq)&" HZ;*WAI"  !set
     marker to Center Freq
1880 OUTPUT @Hp87xx;"SENS1:FREQ:SPAN 200 MHZ;*WAI"  !set Span Freq = 200MHz
1890 !
1900 ! Measure Bandwidth:  Center frequency and -6dB Span frequency
1910 !
1920 OUTPUT @Hp87xx;"ABOR;:INIT1:CONT OFF;:INIT1;*OPC?" !restart sweep
1930 ENTER @Hp87xx;Opc                              !wait for sweep to
     finish
1940 OUTPUT @Hp87xx;"CALC1:MARK:BWID -6;*WAI"       !set bandwidth search to
     -6dB
1950 OUTPUT @Hp87xx;"CALC1:MARK:FUNC BWID;*OPC?"    !search filter for -6dB
     bandwidth
1960 ENTER @Hp87xx;Opc                              !wait for bandwidth to
     be found
1970 OUTPUT @Hp87xx;"CALC1:MARK:FUNC:RES?"          !read the bandwidth data
1980 ENTER @Hp87xx;Freq_span,Freq_center,Q_bpf,Loss_bpf            !read
     in data
1990 OUTPUT @Hp87xx;"CALC1:MARK:AOFF;*WAI"          !markers off
2000 OUTPUT @Hp87xx;"ABOR;:INIT1:CONT ON;*WAI"      !restart continuous
     sweep
2010 !
2020 OUTPUT @Hp87xx;"SENS1:FREQ:SPAN "&VAL$(Freq_span)&" HZ;*WAI"     !set
     Span Freq CH1
2030 OUTPUT @Hp87xx;"SENS1:FREQ:CENT "&VAL$(Freq_center)&" HZ;*WAI"   !set
     Center Freq CH1
2040 OUTPUT @Hp87xx;"SENS2:FREQ:SPAN "&VAL$(Freq_span)&" HZ;*WAI"     !set
     Span Freq CH2
2050 OUTPUT @Hp87xx;"SENS2:FREQ:CENT "&VAL$(Freq_center)&" HZ;*WAI"   !set
     Center Freq CH2
2060 !
2070 !-----------------------------------------------
2080 ! Read marker information (frequency and delay) and display.
2090 !   Note that the X-axis is swept frequency and the Y-axis
2100 !   is delay in seconds.
```

```
2110 !
2120 OUTPUT @Hp87xx;"CALC1:MARK1 ON"         !turn on marker 1 on Y/X trace
2130 OUTPUT @Hp87xx;"CALC1:MARK:FUNC MAX"    !set marker to max
     transmission
2140 OUTPUT @Hp87xx;"CALC2:MARK1 ON"         !turn on marker 1 on AM Delay
     trace
2150 OUTPUT @Hp87xx;"CALC2:MARK1:X?"         !read frequency at marker
2160 ENTER @Hp87xx;Mrkr_freq
2170 OUTPUT @Hp87xx;"CALC2:MARK1:Y?"         !read marker delay value
2180 ENTER @Hp87xx;Mrkr_delay
2190 !
2200 DISP "Marker Frequency = "&VAL$(Mrkr_freq)&" Hz"  !display frequency
2210 WAIT 3
2220 DISP "Marker Delay = "&VAL$(Mrkr_delay)&" Seconds" !display delay
2230 WAIT 3
2240 !
2250 !--------------------------------------------------
2260 ! Read AM Delay trace data, display first data point.
2270 !  Data is transferred in ASCII format for simplicity.
2280 !  Measured data is delay in seconds.  Trace length=201 (default)
2290 !
2300 OUTPUT @Hp87xx;"FORM:DATA ASC,5;:TRAC? CH2FDATA"   !measure formatted
     data
2310 ENTER @Hp87xx;Trace_delay(*)         !read in data
2320 !
2330 DISP "Trace Point #1:  AM Delay = "&VAL$(Trace_delay(1))&" Seconds"
2340 WAIT 3
2350 !
2360 DISP ""                              !clear display line
2370 !
2380 STOP
2390 END
2400 !
2410 !************************************************************
2420 ! Iden_port:   Identify io port to use.
2430 ! Description: This routines sets up the I/O port address for
2440 !              the SCPI interface.  For "HP 87xx" instruments,
2450 !              the address assigned to @Hp87xx = 800 otherwise,
2460 !              716.
2470 !************************************************************
```

```
2480 SUB Iden_port
2490     COM /Sys_state/ @Hp87xx,Scode
2500 !
2510     IF POS(SYSTEM$("SYSTEM ID"),"HP 87")<>0 THEN
2520         ASSIGN @Hp87xx TO 800
2530         Scode=8
2540     ELSE
2550         ASSIGN @Hp87xx TO 716
2560         Scode=7
2570     END IF
2580 !
2590 SUBEND !Iden_port
2600 !
```

9

Front Panel Keycodes

Front Panel Keycodes

Your program can control or monitor the analyzer's front panel with the use of the SCPI `SYSTem:KEY` commands.

Controlling the Front Panel

The front panel can be controlled by sending commands to execute the function of specific keys. The SCPI command `SYSTem:KEY <char>` sends a key name to the analyzer which executes the same function as the corresponding front panel key. For example, `SYSTem:KEY FREQ` will execute the function of the (FREQ) hardkey.

Every hardkey and softkey have a unique key name. Refer to the last table in this chapter for a list of all key names.

Monitoring the Front Panel

The front panel can be monitored to determine when a key has been pressed or when the knob (RPG — rotary pulse generator) has been turned. Key presses from an attached PC DIN keyboard can also be captured.

When keys are pressed or when the knob is turned, the analyzer detects this event, sets bit 0 of the Device Status Register (see Chapter 5, "Using Status Registers") and stores the associated information in a key queue. Your program can use the SCPI `SYSTem:KEY` commands to read the contents of the key queue.

The SCPI query `SYSTem:KEY:TYPE?` returns a string indicating the type of key press event:

Return Value	Meaning
NONE	No key has been pressed
KEY	A front panel key has been pressed
RPG	The analyzer's knob has been turned
ASC	A key on the ASCII PC DIN keyboard has been pressed

The SCPI query `SYSTem:KEY[:VALue]?` returns a number describing the type of key press. The meaning of the number depends on the key type returned by the `SYSTem:KEY:TYPE?` query:

SYST:KEY:TYPE	SYST:KEY:VALUE Meaning
NONE	No meaning. Returns −1.
KEY	A number from 0 to 56 representing the "key code" of the front panel key. See following table for list.
RPG	The number of knob "ticks." Positive values indicate a clock-wise turn; negative numbers indicate counter-clockwise. Larger numbers indicate the knob has been turned faster or further.
ASC	The ASCII value of the pressed key.

The `SYSTem:KEY[:VALue]?` query removes the key from the key queue, so that you can read the next key. For this reason, you must perform the `SYSTem:KEY:TYPE?` query before performing the `SYSTem:KEY[:VALue]?`.

The queue that stores the key press events has a finite length. In firmware revision B.03.00, this length is 32. This means that after 32 key presses occur without being read (using `SYSTem:KEY[:VALue]?`), subsequent key presses or knob ticks will be ignored.

Your program can query the queue length using the SCPI command:

```
SYSTem:KEY:QUEue:MAXimum?
```

You can clear the queue using:

```
SYSTem:KEY:QUEue:CLEar
```

You can check how many key presses or knob tick events have occurred using

```
SYSTem:KEY:QUEUE:COUNt?
```

Finally, you can turn the key queue on or off using

```
SYSTem:KEY:QUEUE[:STATe] <ON|OFF>
```

When the queue is turned off, your program must read each key before a following key is pressed, or information will be lost. It is generally best to leave the queue enabled.

For a complete example of how to read the front panel keys and knob, refer to the KEYCODE example program.

Key Group	Key Label	Key Code	HP-IB Key Name
Softkeys	Softkey 1 (top key)	0	SOFTkey1
	Softkey 2	1	SOFTkey2
	Softkey 3	2	SOFTkey3
	Softkey 4	3	SOFTkey4
	Softkey 5	4	SOFTkey5
	Softkey 6	5	SOFTkey6
	Softkey 7	6	SOFTkey7
	Softkey 8 (bottom key)	7	SOFTkey8
Numeric Keys	(0) (zero)	10	ZERO
	(1) (one)	11	ONE
	(2) (two)	12	TWO
	(3) (three)	13	THRee
	(4) (four)	14	FOUR
	(5) (five)	15	FIVE
	(6) (six)	16	SIX
	(7) (seven)	17	SEVen
	(8) (eight)	18	EIGHt
	(9) (nine)	19	NINE
	(ENTER)	20	ENTer
	(.) (decimal)	21	POINt
	(— / ←) (minus/backspace)	22	MINus
	(⇧) (step up)	23	UP
	(⇩) (step down)	24	DOWN

Key Group	Key Label	Key Code	HP-IB Key Name
Feature Keys	(BEGIN)	40	BEGin
	(MEAS 1)	41	MEAS1
	(MEAS 2)	42	MEAS2
	(POWER)	43	POWer
	(MENU)	44	MENU
	(FREQ)	45	FREQ
	(SWEEP)	46	SWEep
	(CAL)	47	CAL
	(DISPLAY)	48	DISPlay
	(SCALE)	49	SCALe
	(AVG)	50	AVG
	(FORMAT)	51	FORMat
	(MARKER)	52	MARKer
	(SAVE/RECALL)	53	SAVE
	(SYSTEM/OPTIONS)	54	SYSTem
	(HARD/COPY)	55	HARDcopy
	(PRESET)	56	PRESet

10

Introduction to SCPI

Introduction to SCPI

This chapter is a guide to HP-IB control of the analyzer. Its purpose is to provide concise information about the operation of the analyzer under HP-IB control. The reader should already be familiar with making measurements with the analyzer and with the general operation of HP-IB.

Standard Commands for Programmable Instruments (SCPI) is a programming language designed specifically for controlling instruments by Hewlett-Packard and other industry leaders. SCPI provides commands that are common from one instrument to another. This elimination of "device specific" commands for common functions allows programs to be used on different instruments with very little modification.

SCPI was developed to conform to the IEEE 488.2 standard (replacing IEEE 728-1982). The IEEE 488.2 standard defines the syntax and data formats used to send data between devices, the structure of status registers, and the commands used for common tasks. For more information, refer to the IEEE standard itself. SCPI defines the commands used to control device-specific functions, the parameters accepted by these functions, and the values they return.

The Command Tree

The SCPI standard organizes related instrument functions by grouping them together on a common branch of a command tree. Each branch is assigned a mnemonic to indicate the nature of the related functions. The analyzer has 16 major SCPI branches or subsystems. See Figure 10-1 for a model of how these subsystems are organized to manage the measurement and data flow for the analyzer.

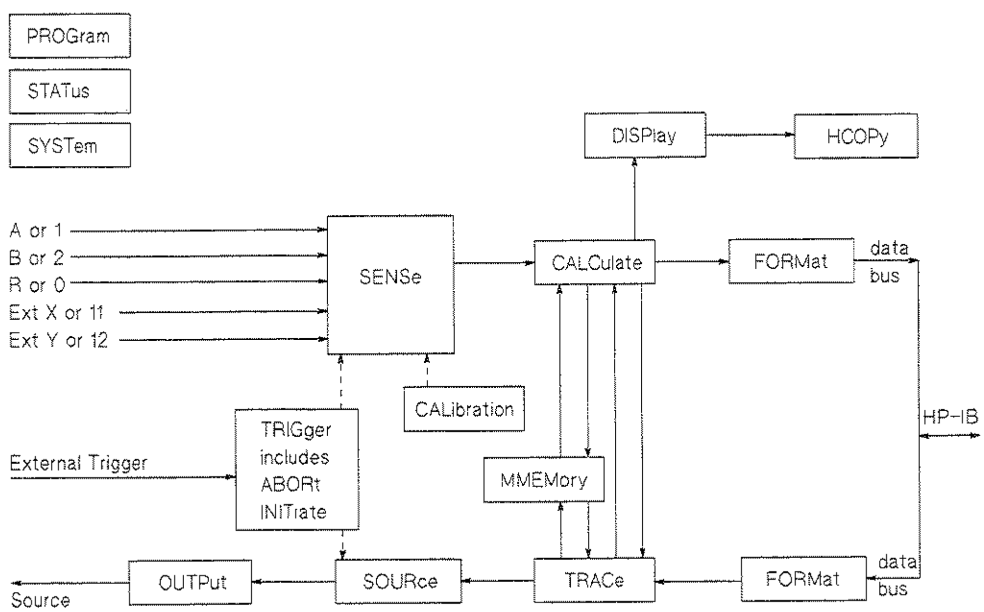


Figure 10-1. Measurement and Data Flow of the Analyzer

The analyzer's major SCPI subsystems and their functions are described below.

`ABORt`	Aborts any sweep in progress.
`CALCulate`	Configures post-measurement processing of the measured data (such as marker and limit testing functions).
`CALibration`	Controls zeroing the broadband diode detectors.
`DISPlay`	Controls the display of measurement data, annotation and user graphics.
`FORMat`	Controls the format of data transfers over the HP-IB. (For more information about HP-IB data transfer refer to Chapter 4, "Data Types and Encoding.")
`HCOPy`	Controls hardcopy (printer and plotter) output.
`INITiate`	Controls the triggering of sweeps.
`MMEMory`	Controls mass storage of instrument states and data (disk and internal memory interface functions).
`OUTPut`	Turns on/off the source output power (power to the device under test).
`PROGram`	Interfaces IBASIC programs and commands with an external controller. (For more information on IBASIC programming refer to *HP Instrument BASIC User's Handbook*.)
`SENSe`	Configures parameters (such as the frequency and measurement parameters) related to the sweep and the measured signal (from the device under test). This subsystem also controls the narrowband calibration routines.
`SOURce`	Controls the RF output power level of the source (power to the device under test).

`STATus`	Contains the commands for using the SCPI status registers. (For more information about using the status registers refer to Chapter 5, "Using Status Registers.")
`SYSTem`	Contains miscellaneous system configuration commands (such as I/O port, clock and softkey control).
`TRACe`	Interfaces with the internal data arrays (functions such as data transfer and trace memory).
`TRIGger`	Controls the source of the sweep triggering.

When many functions are grouped together on a particular branch, additional branching is used to organize these functions into groups that are even more closely related. The branching process continues until each analyzer function is assigned to its own branch. For example, the function that turns on and off the marker tracking feature is assigned to the `TRACKING` branch of the `FUNCTION` branch of the `MARKER` branch of the `CALCULATE` subsystem. The command looks like this:

```
CALCULATE:MARKER:FUNCTION:TRACKING ON
```

NOTE

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

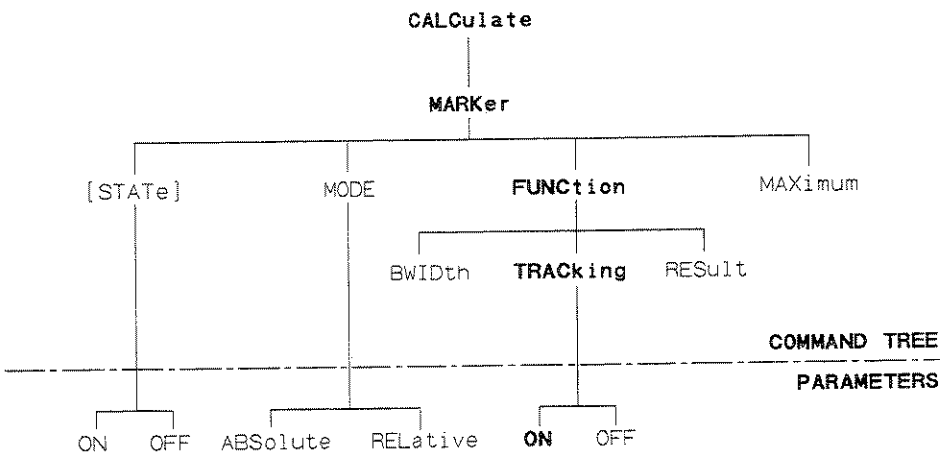


Figure 10-2. Partial Diagram of the CALCulate Subsystem Command Tree

Sending Multiple Commands

Multiple commands can be sent 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 the marker reference and moves the main marker to the highest peak on the trace:

```
OUTPUT 716;"CALCULATE:MARKER:MODE
RELATIVE;:CALCULATE:MARKER:MAXIMUM"
```

One of the analyzer's command parser main functions is to keep track of a program message's position in the command tree. This allows the previous program message to be simplified. Taking advantage of this parser function, the simpler equivalent program message is:

```
OUTPUT 716;"CALCULATE:MARKER:MODE RELATIVE;MAXIMUM"
```

In the first version of 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 base of the command tree. As a result, the next command is only valid if it includes the entire mnemonic path from the base of the tree.

In the second 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 mnemonics of the second command arise from the same branch of the tree as the final mnemonic of the preceding command. MODE, the final mnemonic of the first command, arises from the MARKER branch. So MAXIMUM, the first mnemonic of the second command is also assumed to arise from the MARKER branch.

The following is a longer series of commands – again sent within HP BASIC OUTPUT statements – that can be combined into a single program message:

```
OUTPUT 716;"CALCULATE:MARKER:STATE ON"
OUTPUT 716;"CALCULATE:MARKER:MODE RELATIVE"
OUTPUT 716;"CALCULATE:MARKER:MAXIMUM"
OUTPUT 716;"CALCULATE:MARKER:FUNCTION:TRACKING ON"
```

The single program message is:

```
OUTPUT 716;"CALCULATE:MARKER:STATE ON;MODE
            RELATIVE;MAXIMUM;FUNCTION:TRACKING ON"
```

Command Abbreviation

Each command mnemonic has a long form and a short form. The short forms of the mnemonics allow you to send abbreviated commands. Only the exact short form or the exact long form is accepted.

The short form mnemonics are created according to the following rules:

- If the long form mnemonic has four characters or less, the short form is the same as the long form. For example, `DATA` remains `DATA`.
- If the long form mnemonic 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 mnemonic 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, `LIMIT` becomes `LIM`.

> **NOTE**
>
> The short form of a particular mnemonic is indicated by the use of UPPER-CASE characters in this manual.
>
> SCPI is not case sensitive so any mix of upper- and lower-case lettering can be used when sending commands to the analyzer.

If the rules listed in this section are applied to the last program message in the preceding section, the statement:

```
OUTPUT 716;"CALCULATE:MARKER:STATE ON;MODE
            RELATIVE;MAXIMUM;FUNCTION:TRACKING ON"
```

becomes:

```
OUTPUT 716;"CALC:MARK:STAT ON;MODE REL;MAX;FUNC:TRAC ON"
```

Implied Mnemonics

Some mnemonics can be omitted from HP-IB commands without changing the effect of the command. These special mnemonics are called implied mnemonics, and they are used in many subsystems. In addition to entire mnemonics, variable parts of some mnemonics may also be implied. These are usually a number indicating a particular measurement channel, marker, or similar choice.

NOTE

When a number is not supplied for an implied variable, a default choice is assumed; this choice is always 1.

The `INITIATE` subsystem contains both the implied mnemonic `IMMEDIATE` at its first branching point and an implied variable for the measurement channel. The command to trigger a new sweep is shown in the "SCPI Command Summary" as:

```
OUTPUT 716;"INITiate[1|2][:IMMediate]
```

Any of the following forms of the command can be sent to the analyzer (using HP BASIC) to trigger a new sweep on measurement channel 1:

```
OUTPUT 716;"INITIATE1:IMMEDIATE"
OUTPUT 716;"INITIATE:IMMEDIATE"
OUTPUT 716;"INITIATE1"
OUTPUT 716;"INITIATE"
```

If the sweep is to be triggered for measurement channel 2, the channel number *must* be specified:

```
OUTPUT 716;"INITIATE2:IMMEDIATE"
OUTPUT 716;"INITIATE2"
```

Parameter Types

Parameters are used in many commands. The analyzer uses several types of parameters with different types of commands and queries. When a parameter is sent with a SCPI command it must be separated from the command by a space. If more than one parameter is sent they are separated from each other by commas.

Numeric Parameters

Most subsystems use numeric parameters to specify physical quantities. Simple 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, the instrument automatically rounds the parameter. In addition to numeric values, all numeric parameters accept **MAXimum** and **MINimum** as values (note that **MAXimum** and **MINimum** can be used to set or query values).

<num> is used in this document to denote a numeric parameter.

An example is the command to set the stop frequency for a measurement. The first command below sets the stop frequency to a specific value. The second command below sets the stop frequency to its maximum possible value (1300 MHz for HP 8711C/12C or 3000 MHz for HP 8713C/14C).

```
OUTPUT 716;"SENSE1:FREQUENCY:STOP 1300 MHZ"

OUTPUT 716;"SENSE1:FREQUENCY:STOP MAX"
```

Query Response

When a numeric parameter is queried the number is returned in one of the three numeric formats.

NR1 Integers (such as +1, 0, -1, 123, -12345)

NR2 Floating point number with an explicit decimal point (such as 12.3, +1.234, -0.12345)

NR3 Floating point number in scientific notation (such as +1.23E+5, +123.4E-3, -456.789E+6)

An example is the response to a query of the stop frequency after executing the above commands (this response is of the NR3 type).

```
OUTPUT 716;"SENSE1:FREQUENCY:STOP?"
```

returns the value `1.3E+9`.

Character Parameters

Character parameters (sometimes referred to as discrete parameters) consist of ASCII characters. They are typically used for program settings that have a finite number of values.

These parameters use mnemonics to represent each valid setting. They have a long and a short form which follow the same rules as command mnemonics.

`<char>` is used in this document to denote a character parameter.

An example of a command using a character parameter is the command that selects the format in which the measurement data is displayed:

```
OUTPUT 716;"CALCULATE1:FORMAT MLOGARITHMIC"
```

Query Response

When a character parameter is queried the response is always the short form of the mnemonic that represents the current setting. An example is the response to a query of the data format after executing the above command.

```
OUTPUT 716;"CALCULATE1:FORMAT?"
```

returns the value `MLOG`.

Boolean Parameters

Boolean parameters are used for program settings that can be represented by a single binary condition. Commands that use this type of parameter accept the values ON (or 1) and OFF (or 0).

<ON|OFF> is used in this document to denote a boolean parameter.

An example of a command that uses a boolean parameter is the command that makes the analyzer continuously trigger (or stop triggering) measurements.

```
OUTPUT 716;"INITIATE:CONTINUOUS ON"
```

A special group of commands uses boolean parameters to control automatic functions of the instrument, such as automatically selecting the fastest possible sweep speed. With these automatic functions an additional value is available for the parameter. This value ONCE causes the function to execute once before turning off.

Query Response

The response when a boolean parameter is queried is a single NR1 number indicating the state 1 for on or 0 for off. An example is the response to a query on the sweep trigger status after executing the above command.

```
OUTPUT 716;"INITIATE:CONTINUOUS?"
```

returns the value 1.

String Parameters

String parameters can contain virtually any set of ASCII characters. The string must begin with a single quote (') or a double quote (") and end with the same character (called the delimiter). The delimiter can be included as a character (embedded) inside the string by typing it twice without any characters in between. For example:

```
OUTPUT 716;"DISP:ANN:TITL:DATA 'DUT''S PHASE'"
```

`<string>` is used in this document to denote a string parameter.

A example of a command that uses a string parameter is the `CONFIGURE` command:

```
OUTPUT 716;"CONFIGURE 'FILTER:TRANSMISSION'"
```

Some of the string parameters used by the analyzer, like `'FILTER:TRANSMISSION'` in the example above, follow the same rules that apply to mnemonics. They may have branching (`'FILTER:REFLECTION'` is a related command) and abbreviated versions.

Query Response

The response when a string parameter is queried is a string. The only difference is that the response string will only use double quotes as delimiters. Embedded double quotes may be present in string response data. When the string follows the "SCPI" mnemonic rules, the string returned in response to a query is in the abbreviated form. An example is the response to the configuration status of the analyzer (after executing the last command).

```
OUTPUT 716;"CONFIGURE?"
```

returns the value `"FILT:TRAN"`.

Block Parameters

Block parameters are typically used to transfer large quantities of related data (like a data trace). Blocks can be sent as definite length blocks or indefinite length blocks — the instrument will accept either form. For more information on block data transfers refer to Chapter 4, "Data Types and Encoding."

`<block>` is used in this document to denote a block parameter.

Syntax Summary

The following conventions are used throughout this manual whenever SCPI mnemonics are being described.

angle brackets (< >) — are used to enclose required parameters within a command or query. The definition of the variable is usually explained in the accompanying text.

square brackets ([ ]) — are used to enclose implied or optional parameters within a command or query.

UPPERlower case — are used to indicate the short form (upper-case) of a given mnemonic. The remaining (lower-case) letters are the rest of the long form mnemonic.

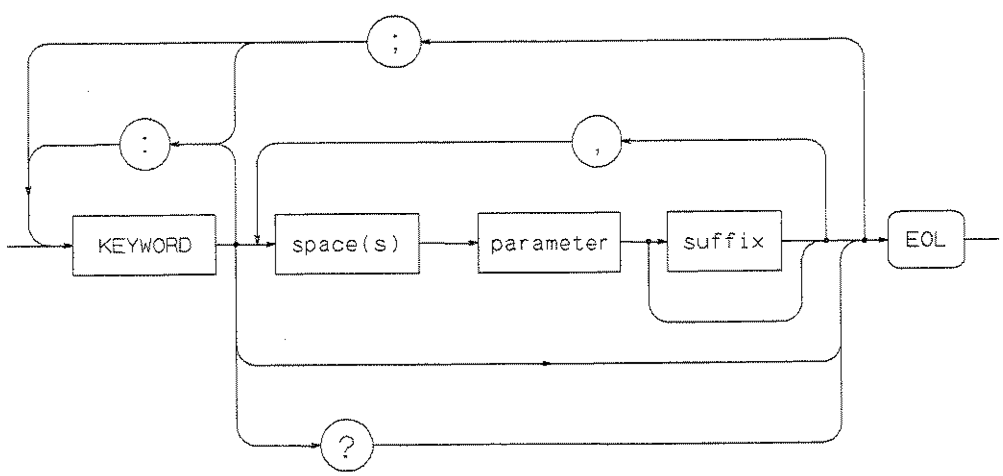


Figure 10-3. SCPI Command Syntax

The following elements have special meanings within a SCPI program message (or combination or mnemonics).

colon (:)	When a command or query contains a series of mnemonics, they are separated by colons. A colon immediately following a mnemonic tells the command parser that the program message is proceeding to the next level of the command tree. A colon immediately following a semicolon tells the command parser that the program message is returning to the base of the command tree.
semicolon (;)	When a program message contains more than one command or query, a semicolon is used to separate them from each other.
comma (,)	A comma separates the data sent with a command or returned with a response.
space ( )	One space is required to separate a command or query from its data (or parameters). Spaces are not allowed inside a command or query.

IEEE 488.2 Common Commands

IEEE 488.2 defines a set of common commands. All instruments are required to implement a subset of these commands, specifically those commands related to status reporting, synchronization and internal operations. The rest of the common commands are optional. The following list details which of these IEEE 488.2 common commands are implemented in the analyzer and the response of the analyzer when the command is received.

`*CLS` Clears the instrument Status Byte by emptying the error queue and clearing all event registers, also cancels any preceding `*OPC` command or query (does not change the enable registers or transition filters).

`*ESE <num>` Sets bits in the Standard Event Status Enable Register — current setting is saved in non-volatile memory.

`*ESE?` Reads the current state of the Standard Event Status Enable Register.

`*ESR?` Reads and clears the current state of the Standard Event Status Register.

`*IDN?` Returns a string that uniquely identifies the analyzer. The string is of the form

```
"HEWLETT-PACKARD,8711C,<serial number>,<software revision>"
```

`*LRN?` This returns a string of device specific characters that, when sent back to the analyzer will restore the instrument state active when `*LRN?` was sent. Data formatting (`ENTER USING "-K"` in HP BASIC) or a similar technique should be used to ensure that the transfer does not terminate on a carriage return or line feed (both $^{C}{}_{R}$ and $^{L}{}_{F}$ are present in the learn string as part of the data).

`*OPC` Operation complete command. The analyzer will generate the `OPC` message in the Standard Event Status Register when all pending overlapped operations have been completed (e.g. a sweep, or a preset). For more information about overlapped operations refer to "Overlapped Commands" in Chapter 2.

`*OPC?` Operation complete query. The analyzer will return an ASCII "1" when all pending overlapped operations have been completed.

`*OPT?` Returns a string identifying the analyzer's option configuration. The string is of the form `"1E1,1C2"`. The options are identified by the following:

1EC	75 ohm
1E1	60 dB step attenuator
1C2	IBASIC
1DA	AM delay (50 Ω)
1DB	AM delay (75 Ω)
1F7	LAN
100	SRL and Fault Location

`*PCB <num>` Sets the pass-control-back address (the address of the controller before a pass control is executed).

`*PSC <num>` Sets the state of the Power-on Status Clear flag — flag is saved in non-volatile memory. This flag determines whether or not the Service Request enable register and the Event Status enable register are cleared at power-up.

`*RST` Executes a device reset and cancels any pending `*OPC` command or query. The contents of the instrument's nonvolatile memory are *not* affected by this command.

This command is different from the front panel (PRESET) function in the state of the commands (and their reset states) listed below.

The preset instrument state is described in the *User's Guide.*

```
INITiate:CONTinuous          = OFF
OUTPut[:STATe]               = OFF
CALibration:ZERO:AUTO        = OFF
SENSe:CORRection[:STATe]     = OFF
SENSe:SWEep:POINts           = MAX
SOURce:POWer                 = MIN
```

`*SRE <num>` Sets bits in the Service Request Enable Register. Current setting is saved in non-volatile memory.

`*SRE?`	Reads the current state of the Service Request Enable Register.
`*STB?`	Reads the value of the instrument Status Byte. This is a non-destructive read, the Status Byte is cleared by the `*CLS` command.
`*TRG`	Triggers a sweep on the active measurement channel when in Trigger Hold mode. Ignored if in continuous sweep.
`*TST?`	Returns the result of a complete self-test. An ASCII `0` indicates no failures found. Any other character indicates a specific self-test failure. Does not perform any self-tests. See the *Service Guide* for further information.
`*WAI`	Prohibits the instrument from executing any new commands until all pending overlapped commands have been completed.

11

Menu Map with SCPI Commands

Menu Map with SCPI Commands

This chapter contains a map of all the softkey menu choices in the analyzer. There is a table for each major hardkey on the analyzer's front panel. The softkeys are shown with corresponding SCPI commands (if one exists). Hardkeys are indicated with the (Hardkey) notation, softkeys are shown as Softkeys. SCPI commands are all shown in their short form.

Some commands (such as source settings) have mnemonics that specify the measurement channel in use. These mnemonics are shown as SENS[1|2]: . . . indicating that either measurement channel could be used. The actual mnemonic entered would be SENS1: ... for setting measurement channel 1 or SENS2: ... for measurement channel 2. Mnemonics for keys that toggle between two states are shown as ... ON|OFF.

<num> and <string> refer to parameter types described in the "Parameter Types" section. <string> parameters are typically enclosed in single quotes ('the string data').

(PRESET) SCPI Command

KEYSTROKES	SCPI COMMAND
(PRESET)	SYST:PRES;*WAI

(BEGIN) SCPI Commands

KEYSTROKES	SCPI COMMAND
(BEGIN)	
Amplifier	
Transmissn	CONF 'AMPL:TRAN';*WAI
Reflection	CONF 'AMPL:REFL';*WAI
Power	CONF 'AMPL:POW';*WAI
Filter	
Transmissn	CONF 'FILT:TRAN';*WAI
Reflection	CONF 'FILT:REFL';*WAI
Broadband Passive	
Transmissn	CONF 'BBAN:TRAN';*WAI
Reflection	CONF 'BBAN:REFL';*WAI
Mixer	
Conversion Loss	CONF 'MIX:CLOS';*WAI
Reflection	CONF 'MIX:REFL';*WAI
AM Delay [1]	CONF 'MIX:GDEL';*WAI

1 Options 1DA and 1DB only

BEGIN SCPI Commands (continued)

KEYSTROKES	SCPI COMMAND
Cable [1]	
Transmissn	CONF[1|2] 'CABL:TRAN';*WAI
Reflection	CONF[1|2] 'CABL:REFL';*WAI
Fault Location	CONF[1|2] 'CABL:FAULT';*WAI
Start Distance	SENS[1|2]:DIST:STAR <num> [FEET|MET];*WAI
Stop Distance	SENS[1|2]:DIST:STOP <num> [FEET|MET];*WAI
Feet	SENS:DIST:UNIT FEET
Meters	SENS:DIST:UNIT MET
Low Pass	SENS:FREQ:MODE LOWP;*WAI
Band Pass	SENS:FREQ:MODE CENT;*WAI
Center Frequency	DISP:ANN:FREQ[1|2]:MODE CSPAN
Number Units	SENS[1|2]:FREQ:CENT <num> [MHZ|KHZ|HZ];*WAI
SRL	CONF[1|2] 'CABL:SRL';*WAI
Start Freq	DISP:ANN:FREQ[1|2]:MODE SSTOP
Number Units	SENS[1|2]:FREQ:STAR <num> [MHZ|KHZ|HZ];*WAI
Stop Freq	DISP:ANN:FREQ[1|2]:MODE SSTOP
Number Units	SENS[1|2]:FREQ:STOP <num> [MHZ|KHZ|HZ];*WAI
Connector Model	
Measure Connector	SENS[1|2]:CORR:MODEL:CONN
Connector Length	
Number ENTER	SENS[1|2]:CORR:LENG:CONN <num>
Connector C Number ENTER	SENS[1|2]:CORR:CAP:CONN <num>

1 Option 100 only

BEGIN SCPI Commands (continued)

KEYSTROKES	SCPI COMMAND
Z Cutoff Frequency	
Number ENTER	SENS:FREQ:ZST <num> [MHZ|KHZ|HZ]
Auto Z ON off	SENS[1|2]:FUNC:SRL:MODE <AUTO|MANUAL>
Manual Z Number ENTER	SENS[1|2]:FUNC:SRL:IMP <num>
SRL Cable Scan	SENS[1|2]:FUNC:SRL:SCAN;*WAI
Number of Points Number ENTER	SENS[1|2]:SWE:POIN <num>;*WAI
User BEGIN[1]	No SCPI command

1 Option 1C2 only

(MEAS 1) | (MEAS 2) SCPI Commands

KEYSTROKES	SCPI COMMAND
(MEAS 1) | (MEAS 2)	SENS[1|2]:STAT ON;*WAI
Transmissn	SENS[1|2]:FUNC 'XFR:POW:RAT 2,0'; DET NBAN;*WAI
Reflection	SENS[1|2]:FUNC 'XFR:POW:RAT 1,0'; DET NBAN;*WAI
Fault Location [1]	SENS[1|2]:FUNC 'FLOC 1,0'; DET NBAN;*WAI
SRL [1]	SENS[1|2]:FUNC 'SRL 1,0'; DET NBAN;*WAI
More	
Power	SENS[1|2]:FUNC 'XFR:POW 2';DET BBAN;*WAI
Conversion Loss	SENS[1|2]:FUNC 'XFR:POW:RAT 2,0'; DET BBAN;*WAI
AM Delay [2]	SENS[1|2]:FUNC 'XFR:GDEL:RAT 12,11'; DET BBAN;*WAI
Detection Options	
Narrowband Internal	
A	SENS[1|2]:FUNC 'XFR:POW 1';DET NBAN;*WAI
B	SENS[1|2]:FUNC 'XFR:POW 2';DET NBAN;*WAI
R	SENS[1|2]:FUNC 'XFR:POW 0';DET NBAN;*WAI
A/R	SENS[1|2]:FUNC 'XFR:POW:RAT 1,0'; DET NBAN;*WAI
B/R	SENS[1|2]:FUNC 'XFR:POW:RAT 2,0'; DET NBAN;*WAI

1 Option 100 only

2 Options 1DA and 1DB only

MEAS 1 | MEAS 2 **SCPI Commands (continued)**

KEYSTROKES	SCPI COMMAND
Broadband Internal	
B*	SENS[1|2]:FUNC 'XFR:POW 2';DET BBAN;*WAI
R*	SENS[1|2]:FUNC 'XFR:POW 0';DET BBAN;*WAI
B*/R*	SENS[1|2]:FUNC 'XFR:POW:RAT 2,0'; DET BBAN;*WAI
Broadband External	
X	SENS[1|2]:FUNC 'XFR:POW 11';DET BBAN;*WAI
Y	SENS[1|2]:FUNC 'XFR:POW 12';DET BBAN;*WAI
X/Y	SENS[1|2]:FUNC 'XFR:POW:RAT 11,12'; DET BBAN;*WAI
Y/X	SENS[1|2]:FUNC 'XFR:POW:RAT 12,11'; DET BBAN;*WAI
Y/R*	SENS[1|2]:FUNC 'XFR:POW:RAT 12,0'; DET BBAN;*WAI
AUX Input	SENS[1|2]:FUNC 'XFR:VOLT';*WAI
Meas OFF	SENS[1|2]:STAT OFF;*WAI
Multiport Selection[1]	
Reflection Port Num	ROUT[1|2]:REFL:PATH:DEF:PORT <1|2| ... |12>
Transmissn Port Num	ROUT[1|2]:TRAN:PATH:DEF:PORT <1|2| ... |12>

1 For use with HP 87075C multiport test sets only.

FREQ SCPI Commands

KEYSTROKES	SCPI COMMAND
FREQ	
Start	DISP:ANN:FREQ[1|2]:MODE SSTOP
Number Units	SENS[1|2]:FREQ:STAR <num> [MHZ|KHZ|HZ];*WAI
Stop	DISP:ANN:FREQ[1|2]:MODE SSTOP
Number Units	SENS[1|2]:FREQ:STOP <num> [MHZ|KHZ|HZ];*WAI
Center	DISP:ANN:FREQ[1|2]:MODE CSPAN
Number Units	SENS[1|2]:FREQ:CENT <num> [MHZ|KHZ|HZ];*WAI
Span	DISP:ANN:FREQ[1|2]:MODE CSPAN
Number Units	SENS[1|2]:FREQ:SPAN <num> [MHZ|KHZ|HZ];*WAI
CW	DISP:ANN:FREQ[1|2]:MODE CW;:SENS[1|2]:FREQ:SPAN 0;*WAI
Number Units	SENS[1|2]:FREQ:CENT <num> [MHZ|KHZ|HZ];*WAI
Fault Loc Frequency[1]	
Low Pass	SENS:FREQ:MODE LOWP;*WAI
Band Pass	SENS:FREQ:MODE CENT;*WAI
Band Pass Max Span Number Units	SENS[1|2]:FREQ:SPAN:MAX <num> [MHZ|KHZ|HZ]

1 Option 100 only

(FREQ) SCPI Commands (continued)

KEYSTROKES	SCPI COMMAND
Disp Freq Resolution	
MHz	DISP:ANN:FREQ:RES MHZ
kHz	DISP:ANN:FREQ:RES KHZ
Hz	DISP:ANN:FREQ:RES HZ

(POWER) SCPI Commands

KEYSTROKES	SCPI COMMAND
(POWER)	
Level (Number) (ENTER)	SOUR[1|2]:POW <num> [dBm];*WAI
RF ON off	OUTP <ON|OFF>;*WAI
Start Power (Number) (ENTER)	SOUR:POW:STAR <num> [dBm];*WAI
Stop Power (Number) (ENTER)	SOUR:POW:STOP <num> [dBm];*WAI
Pwr Sweep Range [1]	
−13 to Max (dBm)	SOUR:POW:RANG ATT0;*WAI
−23 to −8 (dBm)	SOUR:POW:RANG ATT10;*WAI
−33 to −18 (dBm)	SOUR:POW:RANG ATT20;*WAI
−43 to −28 (dBm)	SOUR:POW:RANG ATT30;*WAI
−53 to −38 (dBm)	SOUR:POW:RANG ATT40;*WAI
−60 to −48 (dBm)	SOUR:POW:RANG ATT50;*WAI
−60 to −58 (dBm)	SOUR:POW:RANG ATT60;*WAI
Pwr Level at Preset (Number) (ENTER)	SOUR:POW:PRESET <num> [dBm]

1 The numbers shown on the range keys will depend on the options installed in the analyzer. Also, if the step attenuator option is not installed, these keys will not appear.

(SWEEP) SCPI Commands

KEYSTROKES	SCPI COMMAND
(SWEEP)	
Sweep Time (Number) (ENTER)	SENS[1|2]:SWE:TIME <num> [as|fs|ps|ns|us|ms|s][1];*WAI
Sweep Time AUTO man	SENS[1|2]:SWE:TIME:AUTO <ON|OFF>;*WAI
Alt Sweep on OFF	SENS:COUP <NONE|ALL>;*WAI
Frequency Sweep	POW:MODE:FIX;*WAI
Power Sweep	POW:MODE:SWE;*WAI

1 If using the microsecond suffix ("us"), the letter "u" must be used. Do not use the Greek character "μ."

(MENU) SCPI Commands

KEYSTROKES	SCPI COMMAND
(MENU)	
Trigger	
Continuous	ABOR;:INIT[1|2]:CONT ON;*WAI
Hold	ABOR;:INIT[1|2]:CONT OFF;*WAI
Single	ABOR;:INIT[1|2]:CONT OFF;:INIT[1|2];*WAI
Trigger Source	
Internal	TRIG:SOUR IMM;:SENS:SWE:TRIG:SOUR IMM;*WAI
External Sweep	TRIG:SOUR EXT;:SENS:SWE:TRIG:SOUR IMM;*WAI
External Point	TRIG:SOUR EXT;:SENS:SWE:TRIG:SOUR EXT;*WAI
Number of Points (Number) (ENTER)	SENS[1|2]:SWE:POIN <num>;*WAI
Distance[1]	
Start Distance (Number) (ENTER)	SENS[1|2]:DIST:STAR <num> [FEET|MET];*WAI
Stop Distance (Number) (ENTER)	SENS[1|2]:DIST:STOP <num> [FEET|MET];*WAI
Feet	SENS:DIST:UNIT FEET
Meters	SENS:DIST:UNIT MET
SRL Cable Scan[1]	SENS[1|2]:FUNC:SRL:SCAN;*WAI
Ext Ref on OFF	SENS:ROSC:SOUR <EXT|INT>;*WAI

1 Option 100 only

MENU SCPI Commands (continued)

KEYSTROKES	SCPI COMMAND
Spur Avoid Options	
None	DIAG:SPUR:METH NONE;*WAI
Dither	DIAG:SPUR:METH DITH;*WAI
Spur Avoid	DIAG:SPUR:METH AVO;*WAI

(SCALE) SCPI Commands

KEYSTROKES	SCPI COMMAND
(SCALE)	
Autoscale	DISP:WIND[1|2|10]:TRAC:Y:AUTO ONCE
Scale/Div (Number) (ENTER)	DISP:WIND[1|2]:TRAC:Y:PDIV <num>
Reference Level (Number) (ENTER)	DISP:WIND[1|2]:TRAC:Y:RLEV <num>
Reference Position (Number) (ENTER)	DISP:WIND[1|2]:TRAC:Y:RPOS <num>
Reference Tracking	
Off	DISP:WIND:TRAC[1|2]:Y:TRACK <OFF>
Track Peak	DISP:WIND:TRAC[1|2]:Y:TRACK PEAK
Track Frequency	DISP:WIND:TRAC[1|2]:Y:TRACK FREQ
Set Track Frequency	
(Number) Units	DISP:WIND:TRAC[1|2]:Y:TRACK <num> [MHZ|KHZ|HZ]
⊛ Phase Offset (Number) (ENTER)	SENS[1|2]:CORR:OFFS:PHAS <num> [DEG]
⊛ Electrical Delay (Number) (ENTER)	SENS[1|2]:CORR:EDEL:TIME <num> [as|fs|ps|ns|us|ms|s][1]

1 If using the microsecond unit terminator, the letter "u" must be used. Do not use the Greek character "μ."

(MARKER) SCPI Commands

KEYSTROKES	SCPI COMMAND
(MARKER)	
1: or 1>	CALC[1|2]:MARK1 ON
(Number) Units	CALC[1|2]:MARK1:X <num> [MHZ|KHZ|HZ]
2: or 2>	CALC[1|2]:MARK2 ON
(Number) Units	CALC[1|2]:MARK2:X <num> [MHZ|KHZ|HZ]
3: or 3>	CALC[1|2]:MARK3 ON
(Number) Units	CALC[1|2]:MARK3:X <num> [MHZ|KHZ|HZ]
4: or 4>	CALC[1|2]:MARK4 ON
(Number) Units	CALC[1|2]:MARK4:X <num> [MHZ|KHZ|HZ]
More Markers	
5: or 5>	CALC[1|2]:MARK5 ON
(Number) Units	CALC[1|2]:MARK5:X <num> [MHZ|KHZ|HZ]
6: or 6>	CALC[1|2]:MARK6 ON
(Number) Units	CALC[1|2]:MARK6:X <num> [MHZ|KHZ|HZ]
7: or 7>	CALC[1|2]:MARK7 ON
(Number) Units	CALC[1|2]:MARK7:X <num> [MHZ|KHZ|HZ]
8: or 8>	CALC[1|2]:MARK8 ON
(Number) Units	CALC[1|2]:MARK8:X <num> [MHZ|KHZ|HZ]
Active Marker Off	CALC[1|2]:MARK[1|2|…8] OFF

(MARKER) SCPI Commands (continued)

<table>
<tr><th>KEYSTROKES</th><th>SCPI COMMAND</th></tr>
<tr><td>All Off</td><td>CALC[1|2]:MARK:AOFF</td></tr>
<tr><td>Marker Functions</td><td></td></tr>
<tr><td>   Delta Mkr on OFF</td><td>CALC[1|2]:MARK:MODE <REL|ABS></td></tr>
<tr><td>   Marker -> Center</td><td>SENS[1|2]:FREQ:CENT
(CALC[1|2]:MARK[1|2| ... 8]:X:ABS?);*WAI</td></tr>
<tr><td>   Marker -> Reference</td><td>DISP:WIND[1|2]:TRAC:Y:RLEV
(CALC[1|2]:MARK[1|2| ... 8]:Y?);*WAI</td></tr>
<tr><td>   ⊛ Marker -> Elec Delay</td><td>SENS[1|2]:CORR:EDEL:TIME
(CALC[1|2]:MARK[1|2| ... 8]:GDEL?);*WAI</td></tr>
<tr><td>   Marker Math</td><td></td></tr>
<tr><td>   Statistics</td><td>CALC[1|2]:MARK:FUNC STAT</td></tr>
<tr><td>   Flatness</td><td>CALC[1|2]:MARK:FUNC FLAT</td></tr>
<tr><td>   RF Filter Stats</td><td>CALC[1|2]:MARK:FUNC FST</td></tr>
<tr><td>   Math Off</td><td>CALC[1|2]:MARK:FUNC OFF</td></tr>
<tr><td>Marker Search</td><td></td></tr>
<tr><td>   Max Search</td><td>CALC[1|2]:MARK:FUNC MAX</td></tr>
<tr><td>   Mkr -> Max</td><td>CALC[1|2]:MARK:FUNC MAX</td></tr>
<tr><td>   Next Peak Left</td><td>CALC[1|2]:MARK:MAX:LEFT</td></tr>
<tr><td>   Next Peak Right</td><td>CALC[1|2]:MARK:MAX:RIGH</td></tr>
<tr><td>   Min Search</td><td>CALC[1|2]:MARK:FUNC MIN</td></tr>
<tr><td>   Marker -> Min</td><td>CALC[1|2]:MARK:FUNC MIN</td></tr>
<tr><td>   Next Min Left</td><td>CALC[1|2]:MARK:MIN:LEFT</td></tr>
<tr><td>   Next Min Right</td><td>CALC[1|2]:MARK:MIN:RIGH</td></tr>
</table>

⊛ **indicates HP 8712C/14C only**

(MARKER) SCPI Commands (continued)

KEYSTROKES	SCPI COMMAND
Target Search	CALC[1|2]:MARK:FUNC TARG
Target Value (Number) (ENTER)	CALC[1|2]:MARK:TARG <LEFT|RIGH>,<num> [DB]
Search left (Number) (ENTER)	CALC[1|2]:MARK:TARG LEFT,<num> [DB]
Search right (Number) (ENTER)	CALC[1|2]:MARK:TARG RIGH,<num> [DB]
Bandwidth	CALC[1|2]:MARK:FUNC BWID
(Number) (ENTER)	CALC[1|2]:MARK:BWID <num> [DB]
Notch	CALC[1|2]:MARK:FUNC NOTC
(Number) (ENTER)	CALC[1|2]:MARK:NOTC <num> [DB]
More	
Multi Peak	CALC[1|2]:MARK:FUNC MPE
MultiNotch	CALC[1|2]:MARK:FUNC MNOT
Search Off	CALC[1|2]:MARK:FUNC OFF
Tracking on OFF	CALC[1|2]:MARK:FUNC:TRAC <ON|OFF>

DISPLAY SCPI Commands

KEYSTROKES	SCPI COMMAND
DISPLAY	
Normalize	TRAC CH[1|2]SMEM,CH[1|2]SDATA; :CALC[1|2]:MATH (IMPL/CH[1|2]SMEM); :DISP:WIND[1|2]:TRAC1 ON;TRAC2 OFF
Data->Mem	TRAC CH[1|2]SMEM,CH[1|2]SDATA
Data	CALC[1|2]:MATH (IMPL); :DISP:WIND[1|2]:TRAC1 ON;TRAC2 OFF
Memory	DISP:WIND[1|2]:TRAC1 OFF;TRAC2 ON
Data/Mem	CALC[1|2]:MATH (IMPL/CH[1|2]SMEM); :DISP:WIND[1|2]:TRAC1 ON;TRAC2 OFF
Data and Memory	CALC[1|2]:MATH (IMPL); :DISP:WIND[1|2]:TRAC1 ON;TRAC2 ON
Limit Menu	CALC[1|2]:LIM:DISP ON
Add Limit	
Add Max Line	CALC[1|2]:LIM:SEGM[1|2|···12]:TYPE LMAX; STAT ON
Add Min Line	CALC[1|2]:LIM:SEGM[1|2|···12]:TYPE LMIN; STAT ON
Add Max Point	CALC[1|2]:LIM:SEGM[1|2|···12]:TYPE PMAX; STAT ON
Add Min Point	CALC[1|2]:LIM:SEGM[1|2|···12]:TYPE PMIN; STAT ON
Delete Limit	CALC[1|2]:LIM:SEGM[1|2|···12]:STAT OFF
Delete All Limits	CALC[1|2]:LIM:SEGM:AOFF

(DISPLAY) **SCPI Commands (continued)**

KEYSTROKES	SCPI COMMAND
Edit Limit	
Begin Frequency (Number) (ENTER)	CALC[1|2]:LIM:SEGM[1|2|···12]:FREQ:STAR <num> [MHZ|KHZ|HZ]
End Frequency (Number) (ENTER)	CALC[1|2]:LIM:SEGM[1|2|···12]:FREQ:STOP <num> [MHZ|KHZ|HZ]
Begin Limit (Number) (ENTER)	CALC[1|2|:LIM:SEGM[1|2|···12]:AMPL:STAR <num>
End Limit (Number) (ENTER)	CALC[1|2]:LIM:SEGM[1|2|···12]:AMPL:STOP <num>
Marker	
Limit Options	
Limit Line ON off	CALC[1|2]:LIM:DISP <ON|OFF>
Limit Text ON off	DISP:ANN:LIM:ICON[1|2]:TEXT <ON|OFF>
Limit Icon ON off	DISP:ANN:LIM:ICON[1|2]:FLAG <ON|OFF>
Limit Icon X Position	DISP:ANN:LIM:ICON[1|2]:POS:X <num>
Limit Icon Y Position	DISP:ANN:LIM:ICON[1|2]:POS:Y <num>
Mkr Limits	
Max Limit (Number) (ENTER)	CALC[1|2]:LIM:MARK:STAT :<MEAN|PEAK|FLAT|TILT|FREQ>:MAX <num>
Min Limit (Number) (ENTER)	CALC[1|2]:LIM:MARK:STAT :<MEAN|PEAK|FLAT|TILT|FREQ>:MIN <num>
Mkr Limit ON off	CALC[1|2]:LIM:MARK:STAT :<MEAN|PEAK|FLAT|TILT|FREQ> <ON|OFF>
Limit Test on OFF	CALC[1|2]:LIM:STAT <ON|OFF>

DISPLAY SCPI Commands (continued)

<table>
<tr><th>KEYSTROKES</th><th>SCPI COMMAND</th></tr>
<tr><td>More Display</td><td></td></tr>
<tr><td>Split Display FULL Split</td><td>DISP:FORM [SING|ULOW]</td></tr>
<tr><td>Expand on OFF</td><td>DISP:FORM:EXPAND <ON|OFF></td></tr>
<tr><td>Title and Clock</td><td></td></tr>
<tr><td>Enter Line 1</td><td>DISP:ANN:TITL1:DATA <string></td></tr>
<tr><td>Enter Line 2</td><td>DISP:ANN:TITL2:DATA <string></td></tr>
<tr><td>Show Clock on Line 1</td><td>DISP:ANN:CLOC:MODE LINE1</td></tr>
<tr><td>Show Clock on Line 2</td><td>DISP:ANN:CLOC:MODE LINE2</td></tr>
<tr><td>Clock Off</td><td>DISP:ANN:CLOC:MODE OFF</td></tr>
<tr><td>Title + Clk ON off</td><td>DISP:ANN:TITL <ON|OFF></td></tr>
<tr><td>Color Options</td><td></td></tr>
<tr><td>Factory Default</td><td>DISP:CMAP:SCHEME DEF</td></tr>
<tr><td>Default 2</td><td>DISP:CMAP:SCHEME DEF2</td></tr>
<tr><td>Grey Scale</td><td>DISP:CMAP:SCHEME GREY</td></tr>
<tr><td>Inverse Video</td><td>DISP:CMAP:SCHEME INV</td></tr>
<tr><td>Custom Colors</td><td>DISP:CMAP:SCHEME CUST</td></tr>
<tr><td>Select Item</td><td></td></tr>
<tr><td>Hue, Saturation, Luminance</td><td>DISP:CMAP:COL[1|2| ... 16]:HSL <num>,<num>,<num>[1]</td></tr>
<tr><td>Int Disp Intensity</td><td>DISP:CMAP:COL[1|2| ... 16]:GREY <num></td></tr>
</table>

1 Where **<num>**,**<num>**,**<num>** represents values for hue, saturation, and luminance, respectively. Accepted values for each parameter are 0 to 1.

DISPLAY SCPI Commands (continued) (continued)

KEYSTROKES	SCPI COMMAND
Annotation Options	
Meas Annot ON off	DISP:ANN:CHAN[1|2] <ON|OFF>
Freq Annot ON off	DISP:ANN:FREQ[1|2] <ON|OFF>
Mkr Annot ON off	DISP:ANN:MARK[1|2] <ON|OFF>
Mkr Number ON off	DISP:ANN:MARK[1|2]:NUMB <ON|OFF>
Y-Axis Lbl ON off	DISP:ANN:YAX <ON|OFF>
Y-Axis Lbl rel ABS	DISP:ANN:YAX:MODE <REL|ABS>
Graticule ON off	DISP:WIND[1|2]:TRAC:GRAT:GRID <ON|OFF>

FORMAT SCPI Commands

<table>
<tr><th>KEYSTROKES</th><th>SCPI COMMAND</th></tr>
<tr><td>FORMAT</td><td></td></tr>
<tr><td>Log Mag</td><td>CALC[1|2]:FORM MLOG</td></tr>
<tr><td>Lin Mag</td><td>CALC[1|2]:FORM MLIN</td></tr>
<tr><td>SWR</td><td>CALC[1|2]:FORM SWR</td></tr>
<tr><td>⊛ Delay</td><td>CALC[1|2]:FORM GDEL</td></tr>
<tr><td>⊛ Phase</td><td>CALC[1|2]:FORM PHAS</td></tr>
<tr><td>⊛ Smith Chart</td><td>CALC[1|2]:FORM SMIT</td></tr>
<tr><td>⊛ Polar</td><td>CALC[1|2]:FORM POL</td></tr>
<tr><td>More Format</td><td></td></tr>
<tr><td>⊛ Real</td><td>CALC[1|2]:FORM REAL</td></tr>
<tr><td>⊛ Imaginary</td><td>CALC[1|2]:FORM IMAG</td></tr>
<tr><td>⊛ Impedance Magnitude</td><td>CALC[1|2]:FORM MIMP</td></tr>
<tr><td>Mag Units</td><td></td></tr>
<tr><td>Log Mag Units</td><td>CALC:FORM:UNIT:MLOG
<DBW|DBM|DBUW|DBV|DBMV|DBUV></td></tr>
<tr><td>Lin Mag Units</td><td>CALC:FORM:UNIT:MLIN <W|MW|UW|V|MV|UV></td></tr>
</table>

⊛ indicates HP 8712C/14C only

CAL SCPI Commands

KEYSTROKES	SCPI COMMAND
CAL	
Transmission mode:	
Restore Defaults	SENS[1|2]:CORR:CSET DEF;*WAI
Response	SENS[1|2]:CORR:COLL:IST OFF;METH TRAN1;*WAI
Measure Standard	SENS[1|2]:CORR:COLL STAN1;*WAI; :SENS[1|2]:CORR:COLL:SAVE;*WAI
Response & Isolation	SENS[1|2]:CORR:COLL:IST OFF;METH TRAN2;*WAI
Measure Standard — Loads	SENS[1|2]:CORR:COLL STAN1;*WAI;
Measure Standard — Through	SENS[1|2]:CORR:COLL STAN2;*WAI; :SENS[1|2]:CORR:COLL:SAVE;*WAI
Enhanced Response	SENS[1|2]:CORR:COLL:IST OFF;METH TRAN3;*WAI
Measure Standard — Open	SENS[1|2]:CORR:COLL STAN1;*WAI
Measure Standard — Short	SENS[1|2]:CORR:COLL STAN2;*WAI
Measure Standard — Load	SENS[1|2]:CORR:COLL STAN3;*WAI
Measure Standard — Through	SENS[1|2]:CORR:COLL STAN4;*WAI; :SENS[1|2]:CORR:COLL:SAVE;*WAI

CAL SCPI Commands (continued)

KEYSTROKES	SCPI COMMAND
Transmission mode (continued):	
Test Set Cal [1]	SENS[1|2]:CORR:TESTSET;*WAI
Create "TSET_CAL"	SENS[1|2]:CORR:COLL:METHOD TEST;
XX Ports [2]	SENS[1|2]:CORR:COLL:PORTS <2|4|6|8|10|12>
Measure Opens	SENS[1|2]:CORR:COLL:MP:OPEN <STAN1|STAN2| ... |STAN12>;*WAI;
Measure Shorts	SENS[1|2]:CORR:COLL:MP:SHORT <STAN1|STAN2| ... |STAN12>;*WAI;
Measure Loads	SENS[1|2]:CORR:COLL:MP:LOAD <STAN1|STAN2| ... |STAN12>;*WAI;
Measure Thrus	SENS[1|2]:CORR:COLL:MP:THRU <STAN1|STAN2| ... |STAN6>;*WAI;
All Stds Done	SENS[1|2]:CORR:COLL:SAVE;*WAI;
Abort Cal	SENS[1|2]:CORR:COLL:ABORT
Periodic SelfCal	CAL:SELF <ON|OFF>
SelfCal Once	CAL:SELF ONCE
SelfCal Timer	CAL:SELF:TIMER <num>
SelfCal All Ports	CAL:SELF:ALL

1 For use with HP 87075C multiport test sets only.

2 XX - number of ports

CAL SCPI Commands (continued)

KEYSTROKES	SCPI COMMAND
Transmission mode (continued):	
CAL Check	
Do CAL Check	SENS[1|2]:CORR:COLL:IST OFF;METH VERIFY;*WAI
Measure Standard	SENS[1|2]:CORR:COLL:VER:[TRAN|REFL] <STAN1|STAN2| ... |STAN12>;*WAI;
Abort CAL Check	SENS[1|2]:CORR:COLL:ABORT
View CAL Check	
Directivity	DIAG:MDIS[1|2]:CORR C_DIRECT;*WAI
Source Match	DIAG:MDIS[1|2]:CORR C_SRCMATCH;*WAI
Reflection Tracking	DIAG:MDIS[1|2]:CORR C_RTRACKING;*WAI
Load Match	DIAG:MDIS[1|2]:CORR C_LDMATCH;*WAI
Transmissn Tracking	DIAG:MDIS[1|2]:CORR C_TTRACKING;*WAI
Isolation	DIAG:MDIS[1|2]:CORR C_ISOLATION;*WAI
Restore Meas	DIAG:MDIS[1|2]:REST;*WAI

(CAL) SCPI Commands (continued)

KEYSTROKES	SCPI COMMAND
Reflection mode:	
Restore Defaults	SENS[1|2]:CORR:CSET DEF;*WAI
One Port	SENS[1|2]:CORR:COLL:IST OFF;METH REFL3; *WAI
Measure Standard – Open	SENS[1|2]:CORR:COLL STAN1;*WAI
Measure Standard – Short	SENS[1|2]:CORR:COLL STAN2;*WAI
Measure Standard – Load	SENS[1|2]:CORR:COLL STAN3;*WAI; :SENS[1|2]:CORR:COLL:SAVE;*WAI
Test Set Cal [1]	SENS[1|2]:CORR:TESTSET;*WAI
Create "TSET_CAL"	SENS[1|2]:CORR:COLL:METHOD TEST;
XX Ports [2]	SENS[1|2]:CORR:COLL:PORTS <2|4|6|8|10|12>
Measure Opens	SENS[1|2]:CORR:COLL:MP:OPEN <STAN1|STAN2| ... |STAN12>;*WAI;
Measure Shorts	SENS[1|2]:CORR:COLL:MP:SHORT <STAN1|STAN2| ... |STAN12>;*WAI;
Measure Loads	SENS[1|2]:CORR:COLL:MP:LOAD <STAN1|STAN2| ... |STAN12>;*WAI;
Measure Thrus	SENS[1|2]:CORR:COLL:MP:THRU <STAN1|STAN2| ... |STAN6>;*WAI;
All Stds Done	SENS[1|2]:CORR:COLL:SAVE;*WAI;
Abort Cal	SENS[1|2]:CORR:COLL:ABORT
Periodic SelfCal	CAL:SELF <ON|OFF>
SelfCal Once	CAL:SELF ONCE
SelfCal Timer	CAL:SELF:TIMER <num>
SelfCal All Ports	CAL:SELF:ALL

1 For use with HP 87075C multiport test sets only.

2 XX–number of ports

CAL SCPI Commands (continued) (continued)

KEYSTROKES	SCPI COMMAND
Reflection mode (continued):	
CAL Check	
Do CAL Check	SENS[1|2]:CORR:COLL:IST OFF;METH VERIFY;*WAI
Measure Standard	SENS[1|2]:CORR:COLL:VER:[TRAN|REFL] <STAN1|STAN2| ... |STAN12>;*WAI;
Abort CAL Check	SENS[1|2]:CORR:COLL:ABORT
View CAL Check	
Directivity	DIAG:MDIS[1|2]:CORR C_DIRECT;*WAI
Source Match	DIAG:MDIS[1|2]:CORR C_SRCMATCH;*WAI
Reflection Tracking	DIAG:MDIS[1|2]:CORR C_RTRACKING;*WAI
Load Match	DIAG:MDIS[1|2]:CORR C_LDMATCH;*WAI
Transmissn Tracking	DIAG:MDIS[1|2]:CORR C_TTRACKING;*WAI
Isolation	DIAG:MDIS[1|2]:CORR C_ISOLATION;*WAI
Restore Meas	DIAG:MDIS[1|2]:REST;*WAI

CAL SCPI Commands (continued)

KEYSTROKES	SCPI COMMAND
Fault Location mode:[1]	
Default Cal	SENS[1|2]:CORR:CSET DEF;*WAI
Full Band Cal	SENS[1|2]:CORR:COLL:IST ON;METH REFL3;*WAI
Measure Standard — Open	SENS[1|2]:CORR:COLL STAN1;*WAI
Measure Standard — Short	SENS[1|2]:CORR:COLL STAN2;*WAI
Measure Standard — Load	SENS[1|2]:CORR:COLL STAN3;*WAI; :SENS[1|2]:CORR:COLL:SAVE;*WAI
Velocity Factor	SENS[1|2]:CORR:RVEL:COAX <num>
Cable Loss	SENS[1|2]:CORR:LOSS:COAX <num>
Calibrate Cable	
Specify Length	SENS[1|2]:CORR:LENG:COAX <num> [FEET|MET];*WAI
Measure Cable	SENS[1|2]:CORR:RVEL;*WAI
Multi Peak	
Multi Peak Corr on OFF	SENS[1|2]:CORR:PEAK:COAX [ON|OFF]
Multi Peak Threshold	SENS[1|2]:CORR:THR:COAX <num>
Connector Values	
Connector Length	SENS[1|2]:CORR:LENG:CONN <num>
Connector C	SENS[1|2]:CORR:CAP:CONN <num>

1 Option 100 only.

CAL SCPI Commands (continued)

KEYSTROKES	SCPI COMMAND
SRL mode:[1]	
Default Cal	SENS[1|2]:CORR:CSET DEF;*WAI
Full Band Cal	SENS[1|2]:CORR:COLL:IST ON;METH REFL3;*WAI
Measure Standard — Open	SENS[1|2]:CORR:COLL STAN1;*WAI
Measure Standard — Short	SENS[1|2]:CORR:COLL STAN2;*WAI
Measure Standard — Load	SENS[1|2]:CORR:COLL STAN3;*WAI; :SENS[1|2]:CORR:COLL:SAVE;*WAI
Connector Model	
Measure Connector	
Measure	SENS[1|2]:CORR:MODEL:CONN
Connector Length	SENS[1|2]:CORR:LENG:CONN <num>
Connector C	SENS[1|2]:CORR:CAP:CONN <num>
Z cutoff Frequency	SENS[1|2]:FREQ:ZST <num> [GHZ|MHZ|KHZ|HZ]
Auto Z ON off	SENS[1|2]:FUNC:SRL:MODE [AUTO|MAN]
Manual Z	SENS[1|2]:FUNC:SRL:IMP <num>
AM Delay mode:[2]	
Restore Defaults	SENS[1|2]:CORR:COLL:IST OFF;METH TRAN1;*WAI
Response	SENS[1|2]:CORR:COLL:IST OFF;METH TRAN1;*WAI
Measure Standard	SENS[1|2]:CORR:COLL STAN1;*WAI; :SENS[1|2]:CORR:COLL:SAVE;*WAI

1 Option 100 only.

2 Options 1DA and 1DB only

CAL SCPI Commands (continued)

KEYSTROKES	SCPI COMMAND
Cal Kit	
Default Type-N(f)	SENS:CORR:COLL:CKIT 'COAX,7MM,TYPE-N,50,FEMALE' SENS:CORR:COLL:CKIT 'COAX,7MM,TYPE-N,75,FEMALE' (Option 1EC)
Type-N(m)	SENS:CORR:COLL:CKIT 'COAX,7MM,TYPE-N,50,MALE' SENS:CORR:COLL:CKIT 'COAX,7MM,TYPE-N,75,MALE' (Option 1EC)
User Defined	SENS:CORR:COLL:CKIT 'USER,IMPLIED,IMPLIED, IMPLIED,IMPLIED'
3.5 mm[1]	SENS:CORR:COLL:CKIT 'COAX,3.5MM,APC-3.5,50,IMPLIED'
Type-F[2]	SENS:CORR:COLL:CKIT 'COAX,7MM,TYPE-F,75,IMPLIED'
System Z0	
50 Ω	SENS[1|2]:CORR:IMP:INP:MAGN:SEL Z0_50
75 Ω	SENS[1|2]:CORR:IMP:INP:MAGN:SEL Z0_75
Power mode, Conversion Loss mode, broadband detection modes:	
Autozero	CAL:ZERO:AUTO ON;*WAI
Manual Zero	CAL:ZERO:AUTO ONCE;*WAI
Unratioed narrowband internal measurements:	
Normalize	TRAC CH[1|2]SMEM,CH[1|2]SDATA; :CALC[1|2]:MATH (IMPL/CH[1|2]SMEM); :DISP:WIND[1|2]:TRAC1 ON;TRAC2 OFF

1 50Ω systems only.

2 75Ω systems only.

CAL SCPI Commands (continued)

KEYSTROKES	SCPI COMMAND
⊛ More Cal	
⊛ Velocity Factor (Number) (ENTER)	SENS[1|2]:CORR:RVEL:COAX <num>
⊛ Smith Chart ZO (Number) (ENTER)	SENS[1|2]:CORR:IMP:INP:MAGN <num> [OHM]
⊛ Port Ext's on OFF	SENS[1|2]:CORR:EXT [ON|OFF]
⊛ Refl Port Extension (Number) (ENTER)	SENS[1|2]:CORR:EXT:REFL:TIME <num> [as|fs|ps|ns|us|ms|s][1]
⊛ Trans Port Extension (Number) (ENTER)	SENS[1|2]:CORR:EXT:TRAN:TIME <num> [as|fs|ps|ns|us|ms|s][1]

1 If using the microsecond unit terminator, the letter "u" must be used. Do not use the Greek character "μ."

(AVG) SCPI Commands

KEYSTROKES	SCPI COMMAND
(AVG)	
Average on OFF	SENS[1|2]:AVER <ON|OFF>;*WAI
Restart Average	SENS[1|2]:AVER:CLE;*WAI
Average Factor (Number) (ENTER)	SENS[1|2]:AVER:COUN <num>;*WAI
System Bandwidth	
Wide	SENS[1|2]:BWID 6500 HZ;*WAI
Med Wide	SENS[1|2]:BWID 4000 HZ;*WAI
Medium	SENS[1|2]:BWID 3700 HZ;*WAI
Med Narrow	SENS[1|2]:BWID 1200 HZ;*WAI
Narrow	SENS[1|2]:BWID 250 HZ;*WAI
Fine	SENS[1|2]:BWID 15 HZ;*WAI
Fault Window [1]	
Minimum	SENS[1|2]:WIND RECT
Medium	SENS[1|2]:WIND HAMM
Maximum	SENS[1|2]:WIND KBES
⊛ Delay Aperture	
⊛ Aperture (Hz) (Number) (ENTER)	CALC[1|2]:GDAP:SPAN <num> [HZ];*WAI
⊛ Aperture (%) (Number) (ENTER)	CALC[1|2]:GDAP:APER <num>;*WAI

1 Option 100 only

(SAVE RECALL) SCPI Commands

KEYSTROKES	SCPI COMMAND
(SAVE RECALL)	
Save State	MMEM:STOR:STAT 1,<file>[1]
Re-Save State	MMEM:STOR:STAT 1,<file>[1]
Define Save	
Inst State ON off	MMEM:STOR:STAT:IST <ON|OFF>
Cal on OFF	MMEM:STOR:STAT:CORR <ON|OFF>
Data on OFF	MMEM:STOR:STAT:TRAC <ON|OFF>
TSet Cal on OFF	MMEM:STOR:STAT:TSCAL <ON|OFF>
File Format	
HP 8711A/B Compatible	MMEM:STOR:STAT:FORM HP8711B
HP 8711C Compatible	MMEM:STOR:STAT:FORM HP8711C
Save ASCII	
Lotus 123 Format	MMEM:STOR:TRAC:FORM LOT
Touchstone Format	MMEM:STOR:TRAC:FORM TOUC
Save Meas 1	MMEM:STOR:TRAC CH1FDATA,<file>[1]
Save Meas 2	MMEM:STOR:TRAC CH2FDATA,<file>[1]
Recall State	MMEM:LOAD:STAT 1,<file>[1]

1 **<file>** may include the mass storage device mnemonic **MEM:**, **INT:**, or **RAM:** before the actual name of the file. If the mass storage device is not explicitly named the currently selected device is assumed. **<file>**, **<file1>** and **<file2>** are **<string>** parameters. **<string>** parameters appear between single quotes.

SAVE RECALL **SCPI Commands (continued)**

KEYSTROKES	SCPI COMMAND
Programs	
Save Program	
Re-Save Program	
File Type bin ASCII	
Recall Program	
Save AUTOST	
IBASIC[1]	
Select Disk	
Non-Vol RAM Disk	MMEM:MSIS 'MEM:'
Volatile RAM Disk	MMEM:MSIS 'RAM:'
Internal 3.5" Disk	MMEM:MSIS 'INT:'
Configure VOL_RAM	
Restore Defaults	No SCPI command
Modify Size	No SCPI command
Current Size	No SCPI command

1 The IBASIC menu is described under the **SYSTEM OPTIONS** key.

SAVE RECALL **SCPI Commands (continued)**

KEYSTROKES	SCPI COMMAND
File Utilities [1,2]	
Rename File	MMEM:MOVE <file1>,<file2>
Delete File	MMEM:DEL <file>
Delete All Files	MMEM:DEL '*.*'
Copy File	
Copy to NonVol RAM	MMEM:COPY <file1>,<'MEM:file2'>
Copy to Vol RAM	MMEM:COPY <file1>,<'RAM:file2'>
Copy to 3.5" Int Disk	MMEM:COPY <file1>,<'INT:file2'>
Copy All Files	
Copy to NonVol RAM	MMEM:COPY '*.*','MEM:'
Copy to Vol RAM	MMEM:COPY '*.*','RAM:'
Copy to 3.5" Int Disk	MMEM:COPY '*.*','INT:'
Format Disk	
Format NonVol RAM	MMEM:INIT 'MEM:'
Format Vol RAM	MMEM:INIT 'RAM:'
Format 3.5" Disk	MMEM:INIT 'INT:'

1 Previous models of this analyzer may have supported LIF format. This version only supports DOS.

2 <**file**> may include the mass storage device mnemonic **MEM:**, **INT:**, or **RAM:** before the actual name of the file. If the mass storage device is not explicitly named the currently selected device is assumed. <**file**>, <**file1**> and <**file2**> are <**string**> parameters.

(SAVE RECALL) SCPI Commands (continued)

KEYSTROKES	SCPI COMMAND
Directory Utilities	
Change Directory	MMEM:CDIR <directory>
Make Directory	MMEM:MDIR <directory>
Remove Directory	MMEM:RDIR <directory>
FastRecall on OFF	DISP:MENU:REC:FAST <ON|OFF>

HARD COPY SCPI Commands

KEYSTROKES	SCPI COMMAND
HARD COPY	
Start	HCOP;*WAI
Abort	HCOP:ABOR
Select Copy Port	
Restore Defaults	No SCPI Command
Select	HCOP:DEV:LANG <PCL|HPGL|IBM|EPSON|PCX>; PORT <CENT|SER|GPIB|MMEM|LAN>
LAN Printr IP Addr	SYST:COMM:LAN:PRIN:HOST <string>
Printer Address (Number) (ENTER)	SYST:COMM:GPIB:HCOP:ADDR <num>
Baud Rate (Number) (ENTER)	SYST:COMM:SER:TRAN:BAUD <num>
Xon/Xoff	SYST:COMM:SER:TRAN:HAND XON
DTR/DSR	SYST:COMM:SER:TRAN:HAND DTR
Define PCL5	
Restore Defaults	No SCPI Command
Monochrome	HCOP:DEV3:COL OFF
Color	HCOP:DEV3:COL ON
Auto Feed ON off	HCOP:ITEM3:FFE:STAT <ON|OFF>
Portrait	HCOP:DEV3:PAGE:ORI PORT
Landscape	HCOP:DEV3:PAGE:ORI LAND

HARD COPY SCPI Commands (continued)

KEYSTROKES	SCPI COMMAND
More PCL5	
Restore Defaults	No SCPI Command
Top Margin (Number) (ENTER)	HCOP:DEV3:PAGE:MARG:TOP <num>
Left Margin (Number) (ENTER)	HCOP:DEV3:PAGE:MARG:LEFT <num>
Print Width (Number) (ENTER)	HCOP:DEV3:PAGE:WIDT <num>
Define Printer	
Restore Defaults	No SCPI Command
Monochrome	HCOP:DEV1:COL OFF
Color	HCOP:DEV1:COL ON
Portrait	HCOP:PAGE:ORI PORT
Landscape	HCOP:PAGE:ORI LAND
Auto Feed ON off	HCOP:ITEM1:FFE:STAT <ON|OFF>
More Printer	
Restore Defaults	No SCPI Command
Printer Resolution	
(Number) (ENTER)	HCOP:DEV:RES <num>
Top Margin (Number) (ENTER)	HCOP:PAGE:MARG:TOP <num>
Left Margin (Number) (ENTER)	HCOP:PAGE:MARG:LEFT <num>
Print Width (Number) (ENTER)	HCOP:PAGE:WIDT <num>
Define Plotter	
Restore Defaults	No SCPI Command
Monochrome	HCOP:DEV2:COL OFF
Color	HCOP:DEV2:COL ON

HARD COPY SCPI Commands (continued)

KEYSTROKES	SCPI COMMAND
Set Pen Numbers	
Monochrome Pen (Number) (ENTER)	No SCPI Command
Default Pen Colors	No SCPI Command
Trace 1 Pen (Number) (ENTER)	No SCPI Command
Trace 2 Pen (Number) (ENTER)	No SCPI Command
Memory 1 Pen (Number) (ENTER)	No SCPI Command
Memory 2 Pen (Number) (ENTER)	No SCPI Command
Graticule Pen (Number) (ENTER)	No SCPI Command
Graphics Pen (Number) (ENTER)	No SCPI Command
Auto Feed on OFF	HCOP:ITEM2:FFE:STAT <ON|OFF>
Define Hardcopy	
Restore Defaults	No SCPI Command
Graph and Mkr Table	HCOP:DEV:MODE GMAR
Graph Only	HCOP:DEV:MODE GRAP
Mkr Table Only	HCOP:DEV:MODE MARK
List Trace Values	HCOP:DEV:MODE TABL
Define Graph	
Restore Defaults	No SCPI Command
Trace Data ON off	HCOP:ITEM:TRAC:STAT <ON|OFF>
Graticule ON off	HCOP:ITEM:GRAT:STAT <ON|OFF>
Annotation ON off	HCOP:ITEM:ANN:STAT <ON|OFF>
Mkr Symbol ON off	HCOP:ITEM:MARK:STAT <ON|OFF>
Title + Clk ON off	HCOP:ITEM:TITL:STAT <ON|OFF>

(SYSTEM OPTIONS) SCPI Commands

KEYSTROKES	SCPI COMMAND
(SYSTEM OPTIONS)	
IBASIC[1]	
Run	PROG:STAT RUN
Continue	PROG:STAT CONT
Step	PROG:EXEC 'STEP'
Edit	No SCPI Command
Key Record on OFF	No SCPI Command
Utilities	
Clear Program	PROG:DEL
Stack Size	PROG:MALL <size>
Secure	No SCPI Command
IBASIC Display	
None	DISP:PROG OFF
Full	DISP:PROG FULL
Upper	DISP:PROG UPP
Lower	DISP:PROG LOW

1 Option 1C2 only

(SYSTEM OPTIONS) SCPI Commands (continued)

KEYSTROKES	SCPI COMMAND
LAN [1]	
LAN State on OFF	SYST:COMM:LAN:STAT <ON|OFF>
LAN Port Setup	
HP 8714C IP Address	SYST:COMM:LAN:IPAD <string>
Gateway IP Address	SYST:COMM:LAN:ROUT:GAT <string>
Subnet Mask	SYST:COMM:LAN:ROUT:SMAS <string>
Diagnostic Utilities	
IP Address to Ping	DIAG:COMM:LAN:PING:IPAD <string>
Perform Ping	DIAG:COMM:LAN:PING:IMM
Ethernet Address	SYST:COMM:LAN:EADD?
Security Setup	
HP-IB	
HP 8714C Address (Number) (ENTER)	SYST:COMM:GPIB:ADDR <num> [2]
Talker Listener	SYST:COMM:GPIB:CONT OFF [3]
System Controller	SYST:COMM:GPIB:CONT ON [3]
HP-IB Echo on OFF	SYST:COMM:GPIB:ECHO <ON|OFF>
Operating Parameters	
Hardcopy Screen	No SCPI Command
Hardcopy All	HCOP:DEV:MODE ISET;:HCOP;*WAI
Abort	HCOP:ABOR

1 Option 1F7 only

2 A five second delay is required before a command is sent to the new address.

3 For use with IBASIC running on the analyzer's internal controller — this command cannot be executed from an external controller. Use ***OPC?** and wait for a reply before sending any **OUTPUT 7xx** commands from IBASIC.

SYSTEM OPTIONS SCPI Commands (continued)

KEYSTROKES	SCPI COMMAND
System Config	
Set Clock	
Set Year (Number) (ENTER)	SYST:DATE <year>,<month>,<day>[1]
Set Month (Number) (ENTER)	SYST:DATE <year>,<month>,<day>[1]
Set Day (Number) (ENTER)	SYST:DATE <year>,<month>,<day>[1]
Set Hour (Number) (ENTER)	SYST:TIME <hour>,<minute>,<second>[1]
Set Minute (Number) (ENTER)	SYST:TIME <hour>,<minute>,<second>[1]
Round Seconds	SYST:TIME <hour>,<minute>,0[1]
Clock Format	
YYYY-MM-DD HH:MM	DISP:ANN:CLOC:DATE:FORM YMD
MM-DD-YYYY HH:MM	DISP:ANN:CLOC:DATE:FORM MDY
DD-MM-YYYY HH:MM	DISP:ANN:CLOC:DATE:FORM DMY
Numeric	DISP:ANN:CLOC:DATE:MODE NUM
Alpha	DISP:ANN:CLOC:DATE:MODE ALPH
Seconds ON off	DISP:ANN:CLOC:SEC <ON|OFF>
Done	
Beeper Volume (Number) (ENTER)	SYST:BEEP:VOL <num>[2]

1 <**year**>,<**month**>,<**day**>,<**hour**>,<**minute** > and <**second**> are all <**num**> parameters. Also, these keys do not generate keystroke recording BASIC statements.

2 Number is a fraction, for example 90% would be expressed as 0.90

SYSTEM OPTIONS **SCPI Commands (continued)**

KEYSTROKES	SCPI COMMAND
CRT Adjust	
Restore Defaults	No SCPI Command
Vertical Position	No SCPI Command
Horizontal Position	No SCPI Command
Sync Green on OFF	No SCPI Command
Remove Pattern	No SCPI Command
More	
Restore Defaults	No SCPI Command
Vertical Back Porch	No SCPI Command
Vertical Frnt Porch	No SCPI Command
Horizontal Back Porch	No SCPI Command
Horizontal Frnt Porch	No SCPI Command
Options Setup	
User TTL Config	
Default	SYST:COMM:TTL:USER:FEED DEFAULT
Softkey Auto-Step	SYST:COMM:TTL:USER:FEED KEY
Sweep Out	SYST:COMM:TTL:USER:FEED SWEEP
Switching Test Set [1]	
Multiport on OFF	CONT[1|2]:MULT:STATE <ON|OFF>
Service [2]	

1 For use with HP 87075C multiport test sets only.

2 The Service menu is described in the *Service Guide*.

12

SCPI Command Summary

SCPI Command Summary

This chapter contains all of the HP-IB commands recognized by the analyzer and a brief description. `<num>`, `<char>`, `<string>` and `<block>` refer to the parameter type expected by the instrument as part of the command. All commands have both command and query forms unless specified as command only or query only. Unless otherwise specified, add a "?" to create a query from the command form. For example, the command to select the log magnitude format for the data displayed is `CALCulate[1|2]:FORMat MLOGarithmic`. To query which format is active the corresponding command is `CALCulate[1|2]:FORMat?`. The response to the query is the short form of the mnemonic for the active format, in this example `MLOG`.

The FORM column gives the parameter type returned by the instrument in response to a query. NR1, NR2 and NR3 refer to the different types of numeric data. CHAR (character data), STRING (string data) and BLOCK (block data) are also used to describe response types. These parameter types are described in the "Parameter Types" section of Chapter 10.

Some numeric parameters may be followed by an appropriate suffix. Commands that accept a suffix also allow standard metric multipliers to be combined with the suffix. For example, commands that set a frequency will accept HZ, KHZ, MHZ and GHZ. Commands that set a time will accept S, MS, US, NS, PS, FS and AS. Note that case is ignored. The multiplier "M" is interpreted as either milli or Mega, depending on context. If no suffix is included, the default units for the parameter are used.

NOTE

This SCPI command reference is also available online. It is stored inside your analyzer in electronic form. To access it, you must have the LAN option (1F7). Connect your instrument to the network, and access it using your Web browser. See the *Option 1F7 User's Guide Supplement* for details.

ABORt

SUBSYSTEM COMMANDS	FORM	DESCRIPTION
ABORt	command only	Abort and reset the sweep in progress.

CALCulate

<table>
<tr><th>SUBSYSTEM COMMANDS</th><th>FORM</th><th>DESCRIPTION</th></tr>
<tr><td>CALCulate[1|2]:DATA?[1]</td><td>query only
BLOCK or NR3[2]</td><td>Query the formatted data trace — functionally equivalent to the command TRAC? CH<1|2>FDATA.</td></tr>
<tr><td>CALCulate[1|2]:FORMat <char></td><td>CHAR</td><td>Select the display format for the measurement data — choose from MLOGarithmic|MLINear|SWR
or
⊛–PHASe|SMITh|POLar|GDELay|REAL
|IMAGinary|MIMPedance.</td></tr>
<tr><td>CALCulate[1|2]:FORMat:UNIT:MLIN <char></td><td>CHAR</td><td>Selects linear magnitude units for Y-axis display. Choose from W|MW|UW|V|MV|UV.</td></tr>
<tr><td>CALCulate[1|2]:FORMat:UNIT:MLOG <char></td><td>CHAR</td><td>Selects log magnitude units for Y-axis display. Choose from DBW|DBM|DBUW|DBV|DBMV|DBUV.</td></tr>
<tr><td>⊛CALCulate[1|2]:GDAPerture:APERture <num></td><td>NR3</td><td>Set the group delay aperture as a ratio of desired aperture / measured frequency span.</td></tr>
<tr><td>⊛CALCulate[1|2]:GDAPerture:SPAN <num></td><td>NR3</td><td>Specifies the group delay aperture in Hertz.</td></tr>
<tr><td>CALCulate[1|2]:LIMit:DISPlay <ON|OFF>[3]</td><td>NR1</td><td>Turn on/off display of limit lines.</td></tr>
<tr><td>CALCulate[1|2]:LIMit:MARKer:FLATness :MAXimum <num></td><td>NR3</td><td>Set the maximum value for a flatness limit test.</td></tr>
<tr><td>CALCulate[1|2]:LIMit:MARKer:FLATness :MINimum <num></td><td>NR3</td><td>Set the minimum value for a flatness marker limit test.</td></tr>
<tr><td>CALCulate[1|2]:LIMit:MARKer:FLATness :STATe <ON|OFF>[3]</td><td>NR1</td><td>Turn on/off flatness marker limit test.</td></tr>
</table>

1 Refer to Chapter 6, "Trace Data Transfers," and to the "ASCDATA" and "REALDATA" example programs in Chapter 8 for more information on this command.

2 The parameter type of the data is determined by the format selected — **FORMat REAL** uses BLOCK data, **FORMat ASCii** uses NR3 data separated by commas.

3 Binary parameters accept the values of **1** (on) and **0** (off) in addition to **ON** and **OFF**.

CALCulate (continued)

<table>
<tr><th>SUBSYSTEM COMMANDS</th><th>FORM</th><th>DESCRIPTION</th></tr>
<tr><td>CALCulate[1|2]:LIMit:MARKer:FREQuency :MAXimum <num>[1]</td><td>NR3</td><td>Sets the maximum value for delta frequency marker limit test.</td></tr>
<tr><td>CALCulate[1|2]:LIMit:MARKer:FREQuency :MINimum <num>[1]</td><td>NR3</td><td>Sets the minimum value for delta frequency marker limit test.</td></tr>
<tr><td>CALCulate[1|2]:LIMit:MARKer:FREQuency [:STATe] <ON|OFF>[2]</td><td>NR1</td><td>Turns delta frequency marker limit testing on or off.</td></tr>
<tr><td>CALCulate[1|2]:LIMit:MARKer:STATistic :MEAN:MAXimum <num></td><td>NR3</td><td>Set the maximum value for a statistic mean limit test.</td></tr>
<tr><td>CALCulate[1|2]:LIMit:MARKer:STATistic :MEAN:MINimum <num></td><td>NR3</td><td>Set the minimum value for a statistic mean limit test.</td></tr>
<tr><td>CALCulate[1|2]:LIMit:MARKer:STATistic :MEAN:STATe <ON|OFF>[2]</td><td>NR1</td><td>Turn on/off statistic mean marker limit test.</td></tr>
<tr><td>CALCulate[1|2]:LIMit:MARKer:STATistic :PEAK:MAXimum <num></td><td>NR3</td><td>Set the maximum value for a statistic peak-to-peak limit test.</td></tr>
<tr><td>CALCulate[1|2]:LIMit:MARKer:STATistic :PEAK:MINimum <num></td><td>NR3</td><td>Set the minimum value for a statistic peak-to-peak limit test.</td></tr>
<tr><td>CALCulate[1|2]:LIMit:MARKer:STATistic :PEAK:STATe <ON|OFF>[2]</td><td>NR1</td><td>Turn on/off statistic peak-to-peak marker limit test.</td></tr>
<tr><td>CALCulate[1|2]:LIMit:MARKer:TILT :MAXimum <num></td><td>NR3</td><td>Sets the maximum value for delta amplitude marker limit test.</td></tr>
<tr><td>CALCulate[1|2]:LIMit:MARKer:TILT :MINimum <num>[1]</td><td>NR3</td><td>Sets the minimum value for delta amplitude marker limit test.</td></tr>
<tr><td>CALCulate[1|2]:LIMit:MARKer:TILT [:STATe] <ON|OFF>[2]</td><td>NR1</td><td>Turns delta amplitude marker limit testing on or off.</td></tr>
<tr><td>CALCulate[1|2]:LIMit:SEGMent[1|2| ···12]:AMPLitude:STARt <num>[1]</td><td>NR3</td><td>Set the Begin Limit for the specified limit segment.</td></tr>
</table>

1 Numeric parameters may include an appropriate suffix; if no suffix is included, the default (**HZ** for frequency or **S** for time) is assumed.

2 Binary parameters accept the values of **1** (on) and **0** (off) in addition to **ON** and **OFF**.

CALCulate (continued)

SUBSYSTEM COMMANDS	FORM	DESCRIPTION
CALCulate[1|2]:LIMit:SEGMent[1|2|…12]:AMPLitude:STOP <num>[1]	NR3	Set the End Limit for the specified limit segment.
CALCulate[1|2]:LIMit:SEGMent:AOFF	command only	Turn off all limit segments for a given channel — deletes all segments in the channel's limit table.
CALCulate[1|2]:LIMit:SEGMent[1|2|…12]:DISTance:STARt <num>[1]	NR3	Set the Begin Distance for the specified limit segment. (Option 100 only)
CALCulate[1|2]:LIMit:SEGMent[1|2|…12]:DISTance:STOP <num>[1]	NR3	Set the End Distance for the specified limit segment. (Option 100 only)
CALCulate[1|2]:LIMit:SEGMent[1|2|…12]:FREQuency:STARt <num>[1]	NR3	Set the Begin Frequency for the specified limit segment.
CALCulate[1|2]:LIMit:SEGMent[1|2|…12]:FREQuency:STOP <num>[1]	NR3	Set the End Frequency for the specified limit segment.
CALCulate[1|2]:LIMit:SEGMent[1|2|…12]:POWer:STARt <num>	NR3	Set the Begin Power for the specified limit segment.
CALCulate[1|2]:LIMit:SEGMent[1|2|…12]:POWer:STOP <num>	NR3	Set the End Power for the specified limit segment.
CALCulate[1|2]:LIMit:SEGMent[1|2|…12]:STATe <ON|OFF>[2]	NR1	Turn on/off the specified limit segment — adds or deletes the segment.
CALCulate[1|2]:LIMit:SEGMent[1|2|…12]:TYPE <char>	CHAR	Set the limit type for the specified segment, choose from **LMAX|LMIN|PMAX|PMIN** (Max Line, Min Line, Max Point, Min Point) — sets all of the segment's limit parameters to their default values.
CALCulate[1|2]:LIMit:STATe <ON|OFF>[2]	NR1	Turn on/off the limit test.
CALCulate[1|2]:MARKer:AOFF	command only	Turn off all markers for a given channel — this has the effect of turning off marker functions and tracking as well.
CALCulate[1|2]:MARKer:BWIDth <num>[1]	NR3	Calculate the bandwidth of a bandpass filter — **num** is the target bandwidth (−3 for the 3 dB bandwidth).

1 Numeric parameters may include an appropriate suffix; if no suffix is included, the default (**HZ** for frequency or **S** for time) is assumed.

2 Binary parameters accept the values of **1** (on) and **0** (off) in addition to **ON** and **OFF**.

CALCulate (continued)

SUBSYSTEM COMMANDS	FORM	DESCRIPTION
CALCulate[1|2]:MARKer:FUNCtion :RESult?	query only NR3[,NR3, NR3,NR3]	Query the results of the active marker function — MAX and MIN return the amplitude; TARG returns the frequency; BWID returns bandwidth, center frequency, Q and loss; STAT returns the frequency span, the mean and standard deviation of the amplitude response, and the peak-to-peak ripple; FLAT returns the frequency span, gain, slope and flatness; and FSTAT returns the insertion loss and peak-to-peak ripple of the passband of a filter, as well as the maximum signal amplitude in the stopband. Refer to the "MARKERS" example program in Chapter 8 for more information.
CALCulate[1|2]:MARKer:FUNCtion [:SELect] <char>	CHAR	Select the active marker function — choose from OFF|MAXimum|MINimum|TARGet|BWIDth|NOTCh|MPEak|MNOTCh|STATistics|FLATness|FSTATistics.
CALCulate[1|2]:MARKer:FUNCtion :TRACking <ON|OFF>[1]	NR1	Turn on/off marker function tracking.
CALCulate[1|2]:MARKer[1|2|···8]:GDELay?	query only	Returns the group delay value, in seconds, at the specified marker.
CALCulate[1|2]:MARKer[1|2|···8] :MAXimum	command only	Set the specified marker to the maximum value on the trace.
CALCulate[1|2]:MARKer[1|2|···8] :MAXimum:LEFT	command only	Moves the specified marker to the next local maximum to the left.
CALCulate[1|2]:MARKer[1|2|···8] :MAXimum:RIGHt	command only	Moves the specified marker to the next local maximum to the right.
CALCulate[1|2]:MARKer[1|2|···8] :MINimum	command only	Set the specified marker to the minimum value on the trace.
CALCulate[1|2]:MARKer[1|2|···8] :MINimum:LEFT	command only	Moves the specified marker to the next local minimum to the left.
CALCulate[1|2]:MARKer[1|2|···8] :MINimum:RIGHt	command only	Moves the specified marker to the next local minimum to the right.

1 Binary parameters accept the values of **1** (on) and **0** (off) in addition to **ON** and **OFF**.

CALCulate (continued)

SUBSYSTEM COMMANDS	FORM	DESCRIPTION
CALCulate[1|2]:MARKer:MODE <char>	CHAR	Turn on/off delta marker state — choose ABSolute or RELative.
CALCulate[1|2]:MARKer:NOTCh <num>[1]	NR3	Calculate the notch width of a notch filter — num is the target notch width (-6 for the 6dB bandwidth).
CALCulate[1|2]:MARKer[1|2|···8] :POINt[2]	NR3	Set the specified marker point.
CALCulate[1|2]:MARKer:REFerence:X?	query only NR3	Query the frequency of the reference marker.
CALCulate[1|2]:MARKer:REFerence:Y?	query only NR3	Query the amplitude of the reference marker.
CALCulate[1|2]:MARKer[1|2|···8] [:STATe] <ON|OFF>[3]	NR1	Turn on/off the specified marker.
CALCulate[1|2]:MARKer[1|2|···8] :TARGet <char>,<num>[1]	CHAR,NR3	Perform a marker search for a target value — char is the direction LEFT or RIGHt.
CALCulate[1|2]:MARKer[1|2|···8]:X <num>	NR3	Set the specified marker frequency (or power if in power sweep)
CALCulate[1|2]:MARKer[1|2|···8]:X:ABS <num>	NR?	Set a marker to an absolute value (such as frequency or amplitude). The set value is not relative to a reference marker if one is enabled.
CALCulate[1|2]:MARKer[1|2|···8]:Y?	query only NR3	Query the specified marker amplitude.
⊛CALCulate[1|2]:MARKer[1|2|···8]:Y :INDuctance ?	query only NR3	Query the specified marker's inductance when in Smith chart format.

1 Numeric parameters may include an appropriate suffix; if no suffix is included the default (**HZ** for frequency or **S** for time) is assumed.

2 Refer to "Displaying Measurement Results" in Chapter 7 of the *User's Guide* for more information on using this command.

3 Binary parameters accept the values of **1** (on) and **0** (off) in addition to **ON** and **OFF**.

CALCulate (continued)

SUBSYSTEM COMMANDS	FORM	DESCRIPTION
⊛CALCulate[1|2]:MARKer[1|2|...8]:Y :MAGNitude?	query only NR3	Query the specified marker's magnitude when in polar format.
⊛CALCulate[1|2]:MARKer[1|2|...8]:Y :PHASe?	query only NR3	Query the specified marker's phase value when in polar format.
⊛CALCulate[1|2]:MARKer[1|2|...8]:Y :REACtance?	query only NR3	Query the specified marker's reactance value when in Smith chart format.
⊛CALCulate[1|2]:MARKer[1|2|...8]:Y :RESistance ?	query only NR3	Query the specified marker's resistance value when in Smith chart format.
CALCulate[1|2]:MATH[:EXPRession] <expr>[1]	EXPR[1]	Select a trace math expression — choose measurement trace from (IMPL) for "data only" or (IMPL/CH<1|2>SMEM) for "data / memory".

1 **<expr>** and EXPR represent expressions, a parameter type that consists of mathematical expressions that use character parameters and are enclosed in parentheses.

CALibration

SUBSYSTEM COMMANDS	FORM	DESCRIPTION
`CALibration:SELF:ALL`	command only	Initiates a SelfCal on all ports that were calibrated during the Test Set Cal.[1]
`CALibration:SELF <ON|OFF|ONCE>`[2]	NR1 CHAR	Initiates a SelfCal on the currently selected ports and selects Periodic SelfCal (ON) or SelfCal Once (OFF or ONCE).[1]
`CALibration:SELF:TIMER <num>`	NR1	Sets the time interval for automatic SelfCals.[1]
`CALibration:ZERO:AUTO <ON|OFF|ONCE>`[2]	NR1	Turn on/off the broadband detector autozeroing function.

1 For use with the HP 87075C multiport test set only.

2 Binary parameters accept the values of **1** (on) and **0** (off) in addition to **ON** and **OFF**.

CONFigure

SUBSYSTEM COMMANDS	FORM	DESCRIPTION
`CONFigure <string>`	STRING	Configure the analyzer to measure a specific device type and parameter (the (BEGIN) function) – choose from one of the following strings: 'AMPLifier:TRANsmission' 'AMPLifier:REFLection' 'AMPLifier:POWer' 'FILTer:TRANsmission' 'FILTer:REFLection' 'BBANd:TRANsmission' 'BBANd:REFLection' 'MIXer:CLOSs' 'MIXer:GDEL' 'MIXer:REFLection' 'CABLe:TRANsmission' 'CABLe:REFLection' 'CABLe:FAULT' 'CABLe:SRL'

CONTrol

SUBSYSTEM COMMANDS	FORM	DESCRIPTION
CONTrol[1|2]:MULTiport:STATE <ON|OFF>	NR1	When on, configures analyzer for use with a multiport test set.[1]

1 For use with the HP 87075C multiport test set only.

DIAGnostic

SUBSYSTEM COMMANDS	FORM	DESCRIPTION
DIAGnostic:CCONstants:INSTalled?	query only NR1	Query if correction constants are installed in flash. Returns a 1 if true, and a 0 if false.
DIAGnostic:CCONstants:LOAD	command only	Load default factory calibration constants from floppy disk to memory.
DIAGnostic:CCONstants:STORE:DISK	command only	Store default factory calibration constants from memory to floppy disk.
DIAGnostic:CCONstants:STORE:EEPRom	command only	Store default factory calibration constants from memory to flash EEPROM.
DIAGnostic:COMMunicate:LAN:PING:IMM	command only	Option 1F7 only. "Pings" a remote user-specified IP address. Used in troubleshooting or verifying a LAN connection.
DIAGnostic:COMMunicate:LAN:PING:IPAD <string>	string	Option 1F7 only. Sets the IP address to ping.
DIAGnostic:COMMunicate:LAN:SEND <IP_address>,<port_num>,<string>	NR1 STRING	Instructs the analyzer to open a socket to the specified IP address and port number, and and send the string specified. Must have Option 1F7 (LAN) and Option 1C2 (IBASIC).

DIAGnostic (continued)

<table>
<tr><th>SUBSYSTEM COMMANDS</th><th>FORM</th><th>DESCRIPTION</th></tr>
<tr><td>DIAGnostic:MDISplay[1|2]:CORRection <string></td><td>command only</td><td>Displays corrected measurement uncertainties. Choose from one of the following strings:
Cal check
'C_DIRECTivity'
'C_LDMATCH'
'C_ISOLATION'
'C_RTRACKING'
'C_SRCMATCH'
'C_TTRACKING'
Interpolated Array (accessed through the service menu.)
'I_DIRECTivity'
'I_RESPONSE'
'I_SRCMATCH'
'I_TRACKING'
Master Array (accessed through the service menu.)
'M_DIRECTivity'
'M_RESPONSE'
'M_SRCMATCH'
'M_TRACKING'
'M_XSCALAR'</td></tr>
<tr><td>DIAGnostic:MDISplay[1|2]:RESTore</td><td>command only</td><td>Returns to measurement mode and autoscales after viewing calibration uncertainties.</td></tr>
<tr><td>DIAGnostic:PORT:READ? <port><register>[1]</td><td>query only
NR1, NR1</td><td>Reads the rear panel I/O ports.</td></tr>
<tr><td>DIAGnostic:PORT:WRITE <port><register>[1]</td><td>NR1, NR1, NR1</td><td>Writes to the rear panel I/O ports.</td></tr>
<tr><td>DIAGnostic:SNUMber <string>?</td><td>query only
STRING</td><td>Query the instrument's serial number.</td></tr>
<tr><td>DIAGnostic:SPUR:METHod <NONE|DITHer|AVOid></td><td>NR1</td><td>Select the spur avoid mode.</td></tr>
</table>

1 Refer to "Controlling Peripherals" in Chapter 7 of the *User's Guide* for more information on using this command. Also see tables 12-1 and 12-2.

Table 12-1. Writeable Ports

Port Number	Register	Description
15	0	Outputs 8-bit data to the Cent_D0 through D7 lines of the Centronics port. Cent_D0 is the least significant bit, Cent_D7 is the most significant bit. Checks Centronics status lines for: Out of Paper Printer Not on Line BUSY ACKNOWLEDGE
15	1	Sets/clears the user bit according to the least significant bit of A. A least significant bit equal to 1 sets the user bit high. A least significant bit of 0 clears the user bit.
15	2	Sets/clears the limit pass/fail bit according to the least significant bit of A. A least significant bit equal to 1 sets the pass/fail bit high. A least significant bit of 0 clears the pass/fail bit.
15	3	Outputs 8-bit data to the Cent_D0 through D7 lines of the Centronics port. Cent_D0 is the least significant bit, Cent_D7 is the most significant bit. Does not check Centronics status lines.
9	0	Outputs a byte to the serial port. The byte is output serially according to the configuration for the serial port.

NOTE

When using the `WRITEIO(15,0)` or `WRITEIO(15,3)` command, the Printer_Select Line is set High. However, when the instrument is doing hardcopy, the Printer_Select Line is set low. The Printer_Select line may or may not be used by individual printers. Check with your printer manual.

Table 12-2. Readable Ports

Port Number	Register	Description
9	0	Reads the serial port.
15	0	Reads the 8-bit data port Cent_D0 through D7.
15	1	Reads the user bit.
15	2	Reads the limit test pass/fail bit.
15	10	Reads the 8-bit status port. D0—Cent_acknowledge D1—Cent_busy D2—Cent_out_of_paper D3—Cent_on_line D4—Cent_printer_err

DISPlay

SUBSYSTEM COMMANDS	FORM	DESCRIPTION
DISPlay:ANNotation:CHANnel[1|2] [:STATe] <OFF|ON>[1]	NR1	Enables/disables measurement channel annotation.
DISPlay:ANNotation:CHANnel[1|2]:USER :LABel:DATA <string>[2]	STRING	Specifies the string to be displayed in the measurement channel annotation area (above the graticule).
DISPlay:ANNotation:CHANnel[1|2]:USER :STATe <OFF|ON>[1,2]	NR1	Enables user-defined measurement channel annotation.
DISPlay:ANNotation:CLOCk:DATE :FORMat <char>	CHAR	Selects the Year/Month/Day ordering of the date in the clock display – choose from **YMD|MDY|DMY**.
DISPlay:ANNotation:CLOCk:DATE:MODE <char>	CHAR	Selects the format for the date in the clock display – choose **NUMeric** or **ALPHa**.
DISPlay:ANNotation:CLOCk:MODE <char>	CHAR	Selects how the clock will appear in the measurement display title area – choose from **LINE1|LINE2|OFF**.
DISPlay:ANNotation:CLOCk:SEConds [:STATe] <ON|OFF>[1]	NR1	Turns on/off display of seconds in the clock display.
DISPlay:ANNotation:FREQuency[1|2] :MODE <char>	CHAR	Sets the frequency annotation on the display – choose **SSTOP** (start/stop), **CSPAN** (center/span) or **CW**.
DISPlay:ANNotation:FREQuency[1|2] :RESolution <char>	CHAR	Sets the resolution of display frequency values – choose from **MHZ|KHZ|HZ**.
DISPlay:ANNotation:FREQuency[1|2] [:STATe] <OFF|ON>[1]	NR1	Enables/disables frequency annotation.
DISPlay:ANNotation:FREQuency[1|2] :USER:LABel:DATA <string>	STRING	A user-defined X-axis label.
DISPlay:ANNotation:FREQuency[1|2] :USER:STARt <num>[2]	NR3	Specifies the start value for user-defined frequency annotation.
DISPlay:ANNotation:FREQuency[1|2] :USER:STATe [OFF|ON][1,2]	NR1	Enables user-defined frequency annotation.
DISPlay:ANNotation:FREQuency[1|2] :USER:STOP <num>[2]	NR3	Specifies the stop value for user-defined frequency annotation.

1 Binary parameters accept the values of **1** (on) and **0** (off) in addition to **ON** and **OFF**.

2 Refer to "Displaying measurement Results" in Chapter 7 of the *User's Guide* for more information on using this command.

DISPlay (continued)

<table>
<tr><th>SUBSYSTEM COMMANDS</th><th>FORM</th><th>DESCRIPTION</th></tr>
<tr><td>DISPlay:ANNotation:FREQuency[1|2]:USER:SUFFix[:DATA] <string>[1]</td><td>STRING</td><td>Specifies the suffix for user defined frequency annotation.</td></tr>
<tr><td>DISPlay:ANNotation:LIMit:ICON[1|2]:FLAG[:STATe] <ON|OFF>[2]</td><td>NR1</td><td>Enables/disables the display of the limit test fail icon.</td></tr>
<tr><td>DISPlay:ANNotation:LIMit:ICON[1|2]:POSition:X <num></td><td>NR1</td><td>Positions the limit test pass/fail text and icon on the display. Accepts whole number values from 0 (flush left) to 100 (flush right).</td></tr>
<tr><td>DISPlay:ANNotation:LIMit:ICON[1|2]:POSition:Y <num></td><td>NR1</td><td>Positions the limit test pass/fail text and icon on the display. Accepts whole number values from 0 (bottom of display) to 100 (top of display).</td></tr>
<tr><td>DISPlay:ANNotation:LIMit:ICON[1|2]:TEXT[:STATe] <ON|OFF>[2]</td><td>NR1</td><td>Turns the limit test "PASS/FAIL" text on or off.</td></tr>
<tr><td>DISPlay:ANNotation:MARKer[1|2]:NUMBers[:STATe] <OFF|ON>[2]</td><td>NR1</td><td>Enables/disables the display of marker numbers on trace markers.</td></tr>
<tr><td>DISPlay:ANNotation:MARKer[1|2][:STATe] <ON|OFF>[2]</td><td>NR1</td><td>Enables/disables the active marker annotation for measurement channels 1 and 2.</td></tr>
<tr><td>DISPlay:ANNotation:MESSage:AOFF</td><td>command only</td><td>Turns off any currently showing message window — includes message window, active entry and IBASIC window.</td></tr>
<tr><td>DISPlay:ANNotation:MESSage:CLEar[3]</td><td>command only</td><td>Removes a user-defined pop-up message window.</td></tr>
<tr><td>DISPlay:ANNotation:MESSage:DATA <string>[3]</td><td>STRING</td><td>Displays a user-defined message in the pop-up message window. Optional argument specifies the timeout: choose from NONE|SHORt|MEDium|LONG.</td></tr>
<tr><td>DISPlay:ANNotation:MESSage:STATe <ON|OFF>[2]</td><td>NR1</td><td>Enables/disables the message window — CAUTION: this suppresses display of all messages (even ERROR messages).</td></tr>
<tr><td>DISPlay:ANNotation:TITLe[1|2]:DATA <string>[3]</td><td>STRING</td><td>Enters a string for the specified title line.</td></tr>
<tr><td>DISPlay:ANNotation:TITLe[:STATe] <ON|OFF>[2]</td><td>NR1</td><td>Turns on/off display of the title and clock.</td></tr>
</table>

1 Refer to "Displaying Measurement Results" in Chapter 7 of the *User's Guide* for more information on using this command.

2 Binary parameters accept the values of **1** (on) and **0** (off) in addition to **ON** and **OFF**.

3 Refer to "Operator Interaction" in Chapter 7 of the *User's Guide* for more information on using this command.

DISPlay (continued)

<table>
<tr><th>SUBSYSTEM COMMANDS</th><th>FORM</th><th>DESCRIPTION</th></tr>
<tr><td>DISPlay:ANNotation:YAXis:MODE <char></td><td>CHAR</td><td>Sets mode for the Y-axis labels — choose RELative or ABSolute</td></tr>
<tr><td>DISPlay:ANNotation:YAXis[:STATe] <ON|OFF>[1]</td><td>NR1</td><td>Turns on/off Y-axis labels.</td></tr>
<tr><td>DISPlay:CMAP:COLor[1|2| ... 16] :GREYscale <num></td><td>NR2</td><td>Changes the default intensity of the selected item on the analyzer's internal monitor.</td></tr>
<tr><td>DISPlay:CMAP:COLor[1|2| ... 16]:HSL <num>,<num>,<num></td><td>NR2</td><td>For use with an external VGA compatible monitor. Sets hue, saturation, and luminance for the selected display item. Accepted values for each parameter are 0 to 1.</td></tr>
<tr><td>DISPlay:CMAP:COLor[1|2| ... 16]:RGB <num,num,num></td><td>NR2</td><td>For use with an external monitor. Sets the color map based on the Red/Green/Blue model. Accepted values for each parameter are 0 to 1.</td></tr>
<tr><td>DISPlay:CMAP:DEFault</td><td>command only</td><td>For use with an external monitor. Sets the color scheme to the factory default.</td></tr>
<tr><td>DISPlay:CMAP:SCHeme <char></td><td>CHAR</td><td>Sets the color scheme for an external monitor. Choose from DEFault|DEFault2|GREY|INVerse|CUSTom.</td></tr>
<tr><td>DISPlay:FORMat <char></td><td>CHAR</td><td>Selects the format (full or split screen) for displaying trace data – choose SINGle (overlay) or ULOWer (split).</td></tr>
<tr><td>DISPlay:FORMat:EXPAND <ON|OFF></td><td>NR1</td><td>Enables/disables expand measurement mode.</td></tr>
<tr><td>DISPlay:MENU:KEY[1|2|···7] <string>[2]</td><td>STRING</td><td>Specifies the softkey menu labels from a remote controller or IBASIC</td></tr>
<tr><td>DISPlay:MENU[2]:KEY[1|2|···7] <string>[2]</td><td>STRING</td><td>Specifies the softkey menu labels when using user-defined BEGIN key. (For option 1C2, IBASIC, only)</td></tr>
<tr><td>DISPlay:MENU:RECall:FAST[:STATe] <ON|OFF>[1]</td><td>NR1</td><td>Turns on/off fast recall mode.</td></tr>
<tr><td>DISPlay:PROGram[:MODE] <char></td><td>CHAR</td><td>Selects the portion of the analyzer's screen to be used as an HP Instrument BASIC display — choose from OFF|FULL|UPPer|LOWer.</td></tr>
</table>

1 Binary parameters accept the values of **1** (on) and **0** (off) in addition to **ON** and **OFF**.

2 Refer to "Operator Interaction" in Chapter 7 of the *User's Guide* for more information on using this command.

DISPlay (continued)

<table>
<tr><th>SUBSYSTEM COMMANDS</th><th>FORM</th><th>DESCRIPTION</th></tr>
<tr><td>DISPlay:WINDow[1|2|10]:GEOMetry
:LLEFT?</td><td>query only
NR1,NR1</td><td>Query the absolute pixel coordinates of the lower left corner of the selected display window.</td></tr>
<tr><td>DISPlay:WINDow[1|2|10]:GEOMetry
:SIZE?</td><td>query only
NR1,NR1</td><td>Query the width and height (in pixels) of the selected display window.</td></tr>
<tr><td>DISPlay:WINDow[1|2|10]:GEOMetry
:URIGHT?</td><td>query only
NR1,NR1</td><td>Query the absolute pixel coordinates of the upper right corner of the selected display window.</td></tr>
<tr><td>DISPlay:WINDow:GRAPhics:BUFFer
[:STATe] <ON|OFF>[1]</td><td>NR1</td><td>Turn on/off buffering of user graphics commands.</td></tr>
<tr><td>DISPlay:WINDow[1|2|10]:GRAPhics[2]
:CIRCle <num></td><td>command only</td><td>Draw a circle of the specified Y-axis radius centered at the current pen location — num is the radius in pixels.[3]</td></tr>
<tr><td>DISPlay:WINDow[1|2|10]:GRAPhics[2]
:CLEar</td><td>command only</td><td>Clear the user graphics and graphics buffer for the specified window.</td></tr>
<tr><td>DISPlay:WINDow[1|2|10]:GRAPhics[2]
:COLor <num></td><td>NR1</td><td>Set the color of the user graphics pen — choose from 0 for erase, 1 for bright, and 2 for dim.</td></tr>
<tr><td>DISPlay:WINDow[1|2|10]:GRAPhics[2]
[:DRAW] <num1>,<num2></td><td>command only</td><td>Draw a line from the current pen position to the specified new pen position — num1 and num2 are the new absolute X and Y coordinates in pixels.[3]</td></tr>
<tr><td>DISPlay:WINDow[1|2|10]:GRAPhics[2]
:LABel <string></td><td>command only</td><td>Draw a label with the lower left corner at the current pen location.[3]</td></tr>
<tr><td>DISPlay:WINDow[1|2|10]:GRAPhics[2]
:LABel:FONT <char></td><td>CHAR</td><td>Select the user graphics label font — choose from SMALl|HSMall|NORMal|HNORmal|BOLD|HBOLd|SLANt|HSLant.</td></tr>
<tr><td>DISPlay:WINDow[1|2|10]:GRAPhics[2]
:MOVE <num1>,<num2></td><td>NR1,NR1</td><td>Move the pen to the specified new pen position — num1 and num2 are the new absolute X and Y coordinates in pixels.[3]</td></tr>
<tr><td>DISPlay:WINDow[1|2|10]:GRAPhics[2]
:RECTangle <num1>,<num2></td><td>command only</td><td>Draw a rectangle of the specified size with lower left corner at the current pen position — num1 and num2 are the width and height in pixels.[3]</td></tr>
</table>

1 Binary parameters accept the values of **1** (on) and **0** (off) in addition to **ON** and **OFF**.

2 Refer to Chapter 7, "Using Graphics," for more information.

3 Refer to Chapter 7, and to the example program titled "GRAPHICS" in Chapter 8 for more information.

DISPlay (continued)

<table>
<tr><th>SUBSYSTEM COMMANDS</th><th>FORM</th><th>DESCRIPTION</th></tr>
<tr><td>DISPlay:WINDow[1|2|10]:GRAPhics:SCALe
<xmin>,<xmax>,<ymin,<ymax></td><td>NR1</td><td>Specifies new coordinates for window.</td></tr>
<tr><td>DISPlay:WINDow[1|2|10]:GRAPhics[1]
:STATe?</td><td>query only
NR1</td><td>Query whether a window is enabled for user graphics commands.</td></tr>
<tr><td>DISPlay:WINDow[1|2]:TRACe:
GRATicule:GRID[:STATe] <ON|OFF>[2]</td><td>NR1</td><td>Turn on/off display graticule.</td></tr>
<tr><td>DISPlay:WINDow[1|2]:TRACe[1|2]
[:STATe] <ON|OFF>[2]</td><td>NR1</td><td>Turn on/off the display of trace and memory data from the specified measurement channel.</td></tr>
<tr><td>DISPlay:WINDow[1|2]:TRACe:Y
[:SCALe]:AUTO ONCE</td><td>command only</td><td>Scale the measurement data for a best fit display.</td></tr>
<tr><td>DISPlay:WINDow[1|2]:TRACe:Y
[:SCALe]:PDIVision <num>[3]</td><td>NR3</td><td>Specify the height (dB or units per division) of each vertical division of the specified measurement channel.</td></tr>
<tr><td>DISPlay:WINDow[1|2]:TRACe:Y
[:SCALe]:RLEVel <num>[3]</td><td>NR3</td><td>Specify the value for the Y-axis reference position for the specified measurement channel.</td></tr>
<tr><td>DISPlay:WINDow[1|2]:TRACe:Y
[:SCALe]:RPOSition <num></td><td>NR3</td><td>Specify the Y-axis reference position for the specified measurement channel.</td></tr>
<tr><td>DISPlay:WINDow[1|2|10]:TRACe[1|2]:Y
:TRACk <OFF|PEAK|FREQ></td><td>CHAR</td><td>Selects the method for reference offset tracking.</td></tr>
<tr><td>DISPlay:WINDow[1|2|10]:TRACe[1|2]:Y
:TRACk:FREQuency <num>[3]</td><td>NR3</td><td>Selects frequency to track with reference tracking.</td></tr>
</table>

1 Refer to Chapter 7, "Using Graphics," for more information.

2 Binary parameters accept the values of **1** (on) and **0** (off) in addition to **ON** and **OFF**.

3 Numeric parameters may include an appropriate suffix; if no suffix is included, the default (**HZ** for frequency or **S** for time) is assumed.

FORMat

<table>
<tr><th>SUBSYSTEM COMMANDS</th><th>FORM</th><th>DESCRIPTION</th></tr>
<tr><td>FORMat:BORDer <char></td><td>CHAR</td><td>Specify the byte order used for HPIB data transfer — choose NORMal or SWAPped (for PC-compatible systems).</td></tr>
<tr><td>FORMat[:DATA] <char>[,<num>]</td><td>CHAR[,NR1]</td><td>Specify the data format for use during data transfer — choose from REAL,64|REAL,32|INTeger,16| ASCii.</td></tr>
</table>

HCOPy

<table>
<tr><th>SUBSYSTEM COMMANDS</th><th>FORM</th><th>DESCRIPTION</th></tr>
<tr><td>HCOPy:ABORt</td><td>command only</td><td>Aborts any hardcopy currently in progress.</td></tr>
<tr><td>HCOPy:DEVice[1|2|3]:COLor <ON|OFF>[1,2]</td><td>NR1</td><td>Select monochrome OFF or color ON mode for hardcopy output.</td></tr>
<tr><td>HCOPy:DEVice[1|2|3]:LANGuage <char>[2]</td><td>CHAR</td><td>Select the language for hardcopy output — choose from PCL|HPGL|EPSon|IBM|PCX|PCL5[3]</td></tr>
<tr><td>HCOPy:DEVice[1|2|3]:MODE <char>[2]</td><td>CHAR</td><td>Select the graph and/or table(s) to appear on a hardcopy plot — choose from GMARker|GRAPh|ISETtings|MARKer|TABLe.</td></tr>
<tr><td>HCOPy:DEVice[1|2|3]:PAGE:MARGin:LEFT <num>[2]</td><td>NR2</td><td>Sets the left margin (for printer output) in millimeters.</td></tr>
<tr><td>HCOPy:DEVice[1|2|3]:PAGE:MARGin:TOP <num>[2]</td><td>NR2</td><td>Sets the top margin (for printer output) in millimeters.</td></tr>
<tr><td>HCOPy:DEVice[1|2|3]:PAGE:ORIentation <char>[2]</td><td>CHAR</td><td>Sets printer output page orientation — choose PORTrait or LANDscape.</td></tr>
<tr><td>HCOPy:DEVice[1|2|3]:PAGE:WIDTh <num>[2]</td><td>NR2</td><td>Sets the print width (for printer output) in millimeters.</td></tr>
<tr><td>HCOPy:DEVice[1|2|3]:PORT <char>[2]</td><td>CHAR</td><td>Select the communications port for hardcopy output — choose from CENTronics|SERial|GPIB|MMEMory|LAN.</td></tr>
<tr><td>HCOPy:DEVice[1|2]:RESolution <num>[4]</td><td>NR1</td><td>Sets the printer resolution in dots per inch.</td></tr>
<tr><td>HCOPy[:IMMediate]</td><td>command only</td><td>Initiates a hardcopy output (print or plot).</td></tr>
<tr><td>HCOPy:ITEM[1|2|3]:ANNotation:STATe <ON|OFF>[1,2]</td><td>NR1</td><td>Turns on/off channel and frequency annotation as part of hardcopy output.</td></tr>
<tr><td>HCOPy:ITEM[1|2|3]:FFEed:STATe <ON|OFF>[1,2]</td><td>NR1</td><td>Turns on/off an automatic form feed at the completion of hardcopy output — use item 1 for printers and 2 for plotters.</td></tr>
<tr><td>HCOPy:ITEM[1|2|3]:GRATicule:STATe <ON|OFF>[1,2]</td><td>NR1</td><td>Turns on/off graticule as part of hardcopy output.</td></tr>
</table>

1 Binary parameters accept the values of **1** (on) and **0** (off) in addition to **ON** and **OFF**.

2 For **DEVice**, use **1** for PCL/Epson printers, **2** for plotters, and **3** for PCL5 printers.

3 **EPSon** and **IBM** produce the same results.

4 For **DEVice**, use **1** for PCL/Epson printers, or **2** for plotters.

HCOPy (continued)

<table>
<tr><th>SUBSYSTEM COMMANDS</th><th>FORM</th><th>DESCRIPTION</th></tr>
<tr><td>HCOPy:ITEM[1|2|3]:MARKer:STATe <ON|OFF>[1,2]</td><td>NR1</td><td>Turns on/off marker symbols as part of hardcopy output.</td></tr>
<tr><td>HCOPy:ITEM[1|2|3]:TITLe:STATe <ON|OFF>[1,2]</td><td>NR1</td><td>Turns on/off title and clock lines as part of hardcopy output.</td></tr>
<tr><td>HCOPy:ITEM[1|2|3]:TRACe:STATe <ON|OFF>[1,2]</td><td>NR1</td><td>Turns on/off trace data as part of hardcopy output.</td></tr>
</table>

1 Binary parameters accept the values of **1** (on) and **0** (off) in addition to **ON** and **OFF**.

2 For **DEVice**, use **1** for PCL/Epson printers, **2** for plotters, and **3** for PCL5 printers.

INITiate

<table>
<tr><th>SUBSYSTEM COMMANDS</th><th>FORM</th><th>DESCRIPTION</th></tr>
<tr><td>INITiate[1|2]:CONTinuous <ON|OFF>[1]</td><td>NR1</td><td>Set the trigger system to continuously sweep or to stop sweeping.</td></tr>
<tr><td>INITiate[1|2][:IMMediate]</td><td>command only</td><td>Initiate a new measurement sweep.</td></tr>
</table>

1 Binary parameters accept the values of **1** (on) and **0** (off) in addition to **ON** and **OFF**.

MMEMory

SUBSYSTEM COMMANDS	FORM	DESCRIPTION
MMEMory:CATalog?<string>[1]	query only STRING	List the names of the files in memory.
MMEMory:CDIRectory <string>	STRING	Change the current directory on a DOS formatted disk — new directory must be on the same mass storage device.
MMEMory:COPY <string1>,<string2>[1,2]	command only	Copy a file — **string1** is the source file, **string2** is the destination file.
MMEMory:DELete <string>[1,2]	command only	Delete a file — **string** is the filename.
MMEMory:INITialize [<string>[,<char>[,<num>]]]	command only	Format a disk — **string** is the mass storage device **MEM:** (internal memory), or **INT:** (internal floppy disk. Disk format **char** is **DOS**, and the interleave factor **num**.
MMEMory:LOAD:STATe 1, <string>[1,3]	command only	Recall an instrument state from mass storage — **string** is the filename.
MMEMory:FILE:INFO?<string>[1]	query only STRING	Returns file information such as date/time.
MMEMory:MDIRectory <string>[2]	command only	Make a new directory on a DOS formatted disk.
MMEMory:MOVE <string1>,<string2>[1,2]	command only	Move or rename a file — **string1** is the source (or old) filename and **string2** is the destination (or new) filename.
MMEMory:MSIS <string>	STRING	Select a mass storage device — choose **MEM:** (internal memory), or **INT:** (internal floppy disk drive).
MMEMory:RDIRectory <string>[2]	command only	Delete a directory from a DOS formatted disk.
MMEMory:STORe:STATe 1, <string>[1,2,3]	command only	Save an instrument state to mass storage — **string** is the filename.
MMEMory:STORe:STATe:CORRection <ON|OFF>[4]	NR1	Turn on/off the calibration — part of the definition of a saved file.

1 Filenames may include the mass storage device — **MEM:** (internal non-volatile memory), **RAM:** (internal volatile memory), or **INT:** (internal 3.5" disk drive). Wildcards **?** and * may be used.

2 Be sure to catalog the desired disk using **MMEM:MSIS** before using this command.

3 Refer to "Automated Measurement Setup and Control" in Chapter 7 of the *User's Guide* for more information on using this command.

4 Binary parameters accept the values of **1** (on) and **0** (off) in addition to **ON** and **OFF**.

MMEMory (continued)

<table>
<tr><th>SUBSYSTEM COMMANDS</th><th>FORM</th><th>DESCRIPTION</th></tr>
<tr><td>MMEMory:STORe:STATe:FORMat <char></td><td>CHAR</td><td>Saves instrument state files to be compatible with older "A/B" model analyzers (choose B8711), or with current "C" model analyzers (choose C8711).</td></tr>
<tr><td>MMEMory:STORe:STATe:IState <ON|OFF>[1]</td><td>NR1</td><td>Turn on/off the instrument state — part of the definition of a saved file.</td></tr>
<tr><td>MMEMory:STORe:STATe:TRACe <ON|OFF>[1]</td><td>NR1</td><td>Turn on/off the data trace — part of the definition of a saved file.</td></tr>
<tr><td>MMEMory:STORe:STATe:TSCAL <ON|OFF>[1]</td><td>NR1</td><td>When on, saved state will be the test set cal only.</td></tr>
<tr><td>MMEMory:STORe:TRACe <char>,<string>[2,3]</td><td>command only</td><td>Stores an ASCII list of trace and frequency values to a file — char is the formatted data trace CH<1|2>FDATA and string is the filename.</td></tr>
<tr><td>MMEMory:STORe:TRACe:FORMat <char></td><td>CHAR</td><td>Selects the format that the ASCII data will be saved in. Choose from LOTus123 or TOUChstone.</td></tr>
<tr><td>MMEMory:TRANsfer:BDAT <string>[2][,<block>][4]</td><td>STRING, BLOCK</td><td>Copy a file to or from the analyzer's disk drive.[5]</td></tr>
<tr><td>MMEMory:TRANsfer[:HFS] <string>[2][,<block>][4]</td><td>STRING,</td><td>Copy a file to or from the analyzer's disk drive.</td></tr>
</table>

1 Binary parameters accept the values of **1** (on) and **0** (off) in addition to **ON** and **OFF**.

2 Filenames may include the mass storage device — **MEM:** (internal non-volatile memory), **RAM:** (internal volatile memory), or **INT:** (internal 3.5" disk drive). Wildcards **?** and * may be used.

3 Refer to "Automated Measurement Setup and Control" in Chapter 7 of the *User's Guide* for more information on using this command.

4 Refer to Chapter 8, "Example Programs" for more information on using this command.

5 Refer to the example programs PUTFILE and GETFILE in Chapter 8.

OUTPut

<table>
<tr><th>SUBSYSTEM COMMANDS</th><th>FORM</th><th>DESCRIPTION</th></tr>
<tr><td>OUTPut[:STATe] <ON|OFF>[1]</td><td>NR1</td><td>Turn on/off RF power from the source.</td></tr>
</table>

1 Binary parameters accept the values of **1** (on) and **0** (off) in addition to **ON** and **OFF**.

POWer

SUBSYSTEM COMMANDS	FORM	DESCRIPTION
POWer[1|2]:MODE <char>	CHAR	Specify either frequency sweep (FIXed) or power sweep (SWEep).

PROGram

SUBSYSTEM COMMANDS	FORM	DESCRIPTION
PROGram[1]:CATalog?	query only STRING	List the names of the defined IBASIC programs — response is "PROG" (if a program is present) or the null string ("").
PROGram[1][:SELected][2]:DEFine <block>	BLOCK	Download an IBASIC program from an external controller.
PROGram[1][:SELected][2]:DELete:ALL	command only	Delete all IBASIC programs from the program buffer — equivalent to an HP BASIC SCRATCH A command.
PROGram[1][:SELected][2]:DELete [:SELected]	command only	Delete the active IBASIC program — equivalent to an HP BASIC SCRATCH A command.
PROGram[1][:SELected][2]:EXECute <string>	command only	Execute an IBASIC command.
PROGram[1][:SELected][2]:MALLocate <num>	NR1	Allocate memory space for IBASIC programs — choose from a real number between 2048 and 4000000 bytes.
PROGram[1][:SELected][2]:NAME 'PROG'	STRING	Select the IBASIC program in the program buffer to be active.
PROGram[1][:SELected][2]:NUMBer <string>,<data>[3]	BLOCK or NR3[3]	Load a new value for a numeric variable string in the active IBASIC program — num is the new value.
PROGram[1][:SELected][2]:STATe <char>	CHAR	Select the state of the active IBASIC program — choose from STOP|PAUSe|RUN|CONTinue.
PROGram[1][:SELected][2]:STRing <string1>,<string2>	STRING	Load a new value for a string variable string1 in the active IBASIC program — string2 is the new value.
PROGram[1][:SELected][2]:WAIT	NR1	Wait until the IBASIC program completes.

1 Commands in the **PROGram** subsystem are only available when the HP Instrument BASIC (IBASIC) option is installed (option 1C2). They allow you to generate and control IBASIC programs in the analyzer.

2 Commands grouped under the **SELected** mnemonic in the **PROGram** subsystem operate on the active program buffer.

3 The parameter type of the data is determined by the format selected — **FORMat REAL** uses BLOCK data, **FORMat ASCii** uses NR3 data separated by commas.

ROUTe[1|2]

SUBSYSTEM COMMANDS	FORM	DESCRIPTION
ROUTe[1|2]:REFLection:PATH:DEFine:PORT <1|2| ... |12>	NR1	Selects which port of the test set is connected to the REFLECTION port of the analyzer.[1]
ROUTe[1|2]:TRANsmission:PATH:DEFine: PORT <1|2| ... |12>	NR1	Selects which port of the test set is connected to the TRANSMISSION port of the analyzer.[1]

1 For use with the HP 87075C multiport test set only.

SENSe[1|2]

SUBSYSTEM COMMANDS	FORM	DESCRIPTION
SENSe[1|2]:AVERage:CLEar	command only	Re-start the trace averaging function.
SENSe[1|2]:AVERage:COUNt <num>	NR1	Specify a count or weighting factor for the averaged measurement data.
SENSe[1|2]:AVERage[:STATe] <ON|OFF>[1]	NR1	Turn on/off the trace averaging function.
SENSe[1|2]:BWIDth[:RESolution] <num> HZ	NR2	Specify the bandwidth of the IF receiver (fine, narrow, medium or wide) to be used in the measurement — choose 15 (fine) 250 (narrow) 1200 (medium narrow) 3700 (medium) 4000 (medium wide) or 6500 (wide).
SENSe[1|2]:CORRection:ANNotation?	query only	Returns the current calibration annotation: either "C", "C?", or "".
SENSe[1|2]:CORRection:CAPacitance :CONNector <num>	NR3	Select connector compensating capacitance value. (For use with structural return loss measurements on analyzers with Option 100 only.)
SENSe[1|2]:CORRection:COLLect:ABORt	command only	Aborts the calibration that is currently in progress.
SENSe[1|2]:CORRection:COLLect [:ACQuire] <char>	command only	Measure a calibration standard — select from: STANdard1—Open STANdard2—Short STANdard3—Load STANdard4—Through cable
SENSe[1|2]:CORRection:COLLect :CKIT[:SELect]	STRING	Select Cal Kit Choose from one of the following strings: 'COAX,7MM,TYPE-N,50,FEMALE' 'COAX,7MM,TYPE-N,50,MALE' 'COAX,3.5,APC-3.5,50,IMPLIED' 'USER,IMPLIED,IMPLIED,IMPLIED, IMPLIED' 'COAX,7MM,TYPE-F,75,IMPLIED' 'COAX,7MM,TYPE-N,75,FEMALE' 'COAX,7MM,TYPE-N,75,MALE'
SENSe[1|2]:CORRection:COLLect :IState[:AUTO] <ON|OFF>[1]	NR1	Select the instrument state for calibration — choose Full Band (ON) or User Defined (OFF).

1 Binary parameters accept the values of **1** (on) and **0** (off) in addition to **ON** and **OFF**.

SENSe[1|2] (continued)

SUBSYSTEM COMMANDS	FORM	DESCRIPTION
SENSe[1|2]:CORRection:COLLect :METHod <char>	command only	Select the type of calibration — choose from: TRAN1—Transmission response TRAN2—Transmission response & Isolation TRAN3—Transmission enhanced response REFL3—Reflection one port TEST—Test Set Calibration[1] VERIFY—Calibration Check
SENSe[1|2]:CORRection:COLLect:MP:OPEN <STAN1|STAN2| ... |STAN12>	command only	Measures an open on the port selected during a test set calibration.[1]
SENSe[1|2]:CORRection:COLLect:MP:SHORT <STAN1|STAN2| ... |STAN12>	command only	Measures a short on the port selected during a test set calibration.[1]
SENSe[1|2]:CORRection:COLLect:MP:LOAD <STAN1|STAN2| ... |STAN12>	command only	Measures a load on the port selected during a test set calibration.[1]
SENSe[1|2]:CORRection:COLLect:MP:THRU <STAN1|STAN2| ... |STAN6>	command only	Measures a thru on the port selected during a test set calibration.[1]
SENSe[1|2]:CORRection:COLLect:PORTS <2|4|6|8|10|12>	NR1	Selects the number of ports to perform a test set calibration on.[1]
SENSe[1|2]:CORRection:COLLect:SAVE	command only	Complete and save current calibration.
SENSe[1|2]:CORRection:COLLect:VERify :REFLection <STAN1|STAN2| ... |STAN12>	command only	Measures a calibration standard during a cal check procedure for reflection measurements.
SENSe[1|2]:CORRection:COLLect:VERify :TRANsmission <STAN1|STAN2| ... |STAN12>;	command only	Measures a calibration standard during a cal check procedure for transmission measurements.
SENSe[1|2]:CORRection:CSET [:SELect] DEFault	command only	Restore the "factory" default calibration for the current measurement and channel.
SENSe[1|2]:CORRection:CSET [:SELect]?	query only CHAR	Query the current calibration type — returns DEF (factory default), FULL (full band) or USER (user defined).
⊛SENSe[1|2]:CORRection:EDELay:TIME <num>[2]	NR3	Specifies the electrical delay in seconds.

1 For use with the HP 87075C multiport test set only.

2 Numeric parameters may include an appropriate suffix; if no suffix is included, the default (HZ for frequency or S for time) is assumed.

SENSe[1|2] (continued)

<table>
<tr><th>SUBSYSTEM COMMANDS</th><th>FORM</th><th>DESCRIPTION</th></tr>
<tr><td>⊛SENSe[1|2]:CORRection:EXTension :REFLection[:TIME] <num>[1]</td><td>NR3</td><td>Specifies the port extension at the reflection port, in seconds.</td></tr>
<tr><td>⊛SENSe[1|2]:CORRection:EXTension [:STATe] <ON|OFF>[2]</td><td>NR1</td><td>Enables port extensions.</td></tr>
<tr><td>⊛SENSe[1|2]:CORRection:EXTension :TRANsmission[:TIME] <num>[1]</td><td>NR3</td><td>Specifies the port extension at the transmission port, in seconds.</td></tr>
<tr><td>⊛SENSe[1|2]:CORRection:IMPedance: INPut:MAGNitude<num>[1]</td><td>NR3</td><td>Specifies the reference impedance for the Smith chart display. The default is the analyzer's system impedance.</td></tr>
<tr><td>SENSe[1|2]:CORRection:IMPedance: INPut:MAGNitude:SELect ZO_50</td><td>NR1</td><td>Selects 50 ohms as the system impedance.</td></tr>
<tr><td>SENSe[1|2]:CORRection:IMPedance: INPut:MAGNitude:SELect ZO_75</td><td>NR1</td><td>Selects 75 ohms as the system impedance.</td></tr>
<tr><td>SENSe[1|2]:CORRection:LENGth:COAX <num></td><td>NR2</td><td>Specifies the length of cable to be calibrated, in feet or meters. (For use with fault location measurements on analyzers with Option 100 only.)</td></tr>
<tr><td>SENSe[1|2]:CORRection:LENGth :CONNector <num></td><td>NR2</td><td>Specifies the length of an interface connector, in mm or inches. (For use with structural return loss measurements on analyzers with Option 100 only.)</td></tr>
<tr><td>SENSe[1|2]:CORRection:LOSS:COAX <num></td><td>NR2</td><td>Specifies the loss of a cable under test, in dB/100 ft. (For use with fault location measurements on analyzers with Option 100 only.)</td></tr>
<tr><td>SENSe[1|2]:CORRection:MODel:CONNector [:IMMediate]</td><td>command only</td><td>Measure the cable connector and determine the optimum values for connector length and connector capacitance. (For use with structural return loss measurements on analyzers with Option 100 only.)</td></tr>
</table>

1 Numeric parameters may include an appropriate suffix; if no suffix is included, the default (**HZ** for frequency or **S** for time) is assumed.

2 Binary parameters accept the values of **1** (on) and **0** (off) in addition to **ON** and **OFF**.

SENSe[1|2] (continued)

<table>
<tr><th>SUBSYSTEM COMMANDS</th><th>FORM</th><th>DESCRIPTION</th></tr>
<tr><td>⊛SENSe[1|2]:CORRection:OFFSet:PHASe</td><td>NR3</td><td>Specifies the phase offset.</td></tr>
<tr><td>SENSe[1|2]:CORRection:PEAK:COAX [:STATe] <ON|OFF>[1]</td><td>NR1</td><td>Turns multi-peak correction on or off. (For use with fault location measurements on analyzers with Option 100 only.)</td></tr>
<tr><td>⊛SENSe[1|2]:CORRection:RVELocity:COAX <num>[2]</td><td>NR3</td><td>Specifies the velocity factor to be used when displaying the distance for electrical length and port extensions. 1.0 = the speed of light.</td></tr>
<tr><td>SENSe[1|2]:CORRection:RVELocity [:IMMediate]</td><td>command only</td><td>Measure the cable and determine the optimum values for cable loss and velocity factor. (For use with fault location measurements on analyzers with Option 100 only.)</td></tr>
<tr><td>SENSe[1|2]:CORRection:TESTSET</td><td>command only</td><td>Brings up the Test Set Cal menu.[3]</td></tr>
<tr><td>SENSe[1|2]:CORRection:THReshold :COAX <num></td><td>NR2</td><td>Selects multi-peak threshold value, in dB. (For use with fault location measurements on analyzers with Option 100 only.)</td></tr>
<tr><td>SENSe[1|2]:COUPle <char></td><td>CHAR</td><td>Turn on/off the alternate sweep mode — choose ALL (coupled sweep) or NONE (alternate sweep).</td></tr>
<tr><td>SENSe[1|2]:DETector[:FUNCtion] <char></td><td>CHAR</td><td>Specify which detection mode is used to make the measurement — choose BBANd (broadband) or NBANd (narrowband).</td></tr>
<tr><td>SENSe[1|2]:DISTance:CENTer <num></td><td>NR3</td><td>Set the center distance for a fault location measurement, in feet or meters. (For use with fault location measurements on analyzers with Option 100 only.)</td></tr>
</table>

1 Binary parameters accept the values of **1** (on) and **0** (off) in addition to **ON** and **OFF**.

2 Numeric parameters may include an appropriate suffix; if no suffix is included the default (**HZ** for frequency or **S** for time) is assumed.

3 For use with the HP 87075C multiport test set only.

SENSe[1|2] (continued)

SUBSYSTEM COMMANDS	FORM	DESCRIPTION
SENSe[1|2]:DISTance:STARt <num>	NR3	Set the start distance for a fault location measurement, in feet or meters. (For use with fault location measurements on analyzers with Option 100 only.)
SENSe[1|2]:DISTance:STOP <num>	NR3	Set the stop distance for a fault location measurement, in feet or meters. (For use with fault location measurements on analyzers with Option 100 only.)
SENSe[1|2]:DISTance:UNITs <char>	CHAR	Specifies distance units. Choose METers or FEET. (For use with fault location measurements on analyzers with Option 100 only.)
SENSe[1|2]:FREQuency:CENTer <num>[1]	NR3	Set the center frequency of the RF source.
SENSe[1|2]:FREQuency:MODE <char>	CHAR	Set the fault location measurement to CENTer (bandpass) or LOWPass.(For use with fault location measurements on analyzers with Option 100 only.)
SENSe[1|2]:FREQuency:SPAN <num>[1]	NR3	Set the frequency span of the RF source.
SENSe[1|2]:FREQuency:SPAN :MAXimum <num>[1]	NR3	Set the maximum frequency span of the RF source for bandpass fault location measurements. (For use with fault location measurements on analyzers with Option 100 only.)
SENSe[1|2]:FREQuency:STARt <num>[1]	NR3	Set the start frequency of the RF source.
SENSe[1|2]:FREQuency:STOP <num>[1]	NR3	Set the stop frequency of the RF source.
SENSe[1|2]:FREQuency:ZSTOp <num>[1]	NR3	Set the Z cutoff frequency for cable impedance calculations. (For use with structural return loss measurements on analyzers with Option 100 only.)
SENSe[1|2]:FUNCtion?	query only STRING	Query the measurement function — returns one of the 'XFR:POW . . .' or 'XFR:POW:RAT ... ' strings described below.
SENSe[1|2]:FUNCtion 'XFRequency :POWer <num>'	command only	Specify that the receiver will measure the power into a specific detector on the specified measurement channel — choose from detectors 0 (R), 1 (A), 2 (B), 11 (Ext X) or 12 (Ext Y).

1 Numeric parameters may include an appropriate suffix; if no suffix is included, the default (**HZ** for frequency or **S** for time) is assumed.

SENSe[1|2] (continued)

<table>
<tr><th>SUBSYSTEM COMMANDS</th><th>FORM</th><th>DESCRIPTION</th></tr>
<tr><td>SENSe[1|2]:FUNCtion 'XFRequency :POWer:RATio <num>,<num>'</td><td>command only</td><td>Specify that the receiver will measure a ratio of the power into the specified measurement channel — choose from ratios 1,0 (A/R), 2,0 (B/R), 12,0 (Ext Y/R), 11,12 (Ext X/Ext Y), or 12,11 (Ext Y/Ext X).</td></tr>
<tr><td>SENSe[1|2]:FUNCtion:FAULt:CONNector [:IMMediate]</td><td>command only</td><td>Forces a connector verification measurement on the alternate channel. (For use with fault location measurement on analyzers with Option 100 only.)</td></tr>
<tr><td>SENSe[1|2]:FUNCtion:SRL:IMPedance <num></td><td>NR2</td><td>Set the cable impedance. (For use with structural return loss measurements on analyzers with Option 100 only.)</td></tr>
<tr><td>SENSe[1|2]:FUNCtion:SRL:MODE <char></td><td>CHAR</td><td>Set the auto z function to AUTO or MANual. (For use with structural return loss measurements on analyzers with Option 100 only.)</td></tr>
<tr><td>SENSe[1|2]:FUNCtion:SRL:SCAN [:IMMediate]</td><td>command only</td><td>Start a cable scan. (For use with structural return loss measurements on analyzers with Option 100 only.)</td></tr>
<tr><td>SENSe[1|2]:ROSCillator:SOURce <char></td><td>CHAR</td><td>Specify the source of the reference oscillator — select INTernal or EXTernal.</td></tr>
<tr><td>SENSe[1|2]:STATe <ON|OFF>[1]</td><td>NR1</td><td>Turn on/off the specified channel.</td></tr>
<tr><td>SENSe[1|2]:SWEep:POINts <num></td><td>NR1</td><td>Set the number of data points for the measurement — choose from 3|5|11|21|51|101|201|401|801|1601.</td></tr>
<tr><td>SENSe[1|2]:SWEep:TIME <num>[2]</td><td>NR3</td><td>Set the sweep time.</td></tr>
<tr><td>SENSe[1|2]:SWEep:TIME:AUTO <ON|OFF|ONCE>[1]</td><td>NR1</td><td>Turn on/off the automatic sweep time function.</td></tr>
<tr><td>SENSe:SWEep:TRIGger:SOURce <char></td><td>CHAR</td><td>Set the trigger source for each point in a sweep — choose IMMediate or EXTernal (used in conjunction with TRIGger[:SEQuence]:SOURce).</td></tr>
<tr><td>SENSe:WINDow[:TYPE] <char></td><td>CHAR</td><td>Set the window selection for fault location measurements. Choose from RECTangular (Minimum), HAMMing (Medium), or KBESsel (Maximum). (For use with fault location measurements on analyzers with Option 100 only.)</td></tr>
</table>

1 Binary parameters accept the values of **1** (on) and **0** (off) in addition to **ON** and **OFF**.

2 Numeric parameters may include an appropriate suffix; if no suffix is included, the default (**HZ** for frequency or **S** for time) is assumed.

SOURce

SUBSYSTEM COMMANDS	FORM	DESCRIPTION
SOURce[1|2]:POWer[:LEVel] [:IMMediate][:AMPLitude] <num>[1]	NR3	Set the RF power output from the source.
SOURce[1|2]:POWer:PRESet <num>	NR3	Sets the power level that the analyzer will always return to after an instrument preset.
SOURce[1|2]:POWer:RANGe <char>	CHAR	Specifies the power sweep range. Choose from ATTen0 |ATTen10|ATTen20|ATTen30|ATTen40 |ATTen50|ATTen60.
SOURce[1|2]:POWer:STARt <num>	NR3	Sets the power sweep start power.
SOURce[1|2]:POWer:STOP <num>	NR3	Sets the power sweep stop power.

1 Numeric parameters may include an appropriate suffix; if no suffix is included, the default (**HZ** for frequency or **S** for time) is assumed.

STATus

SUBSYSTEM COMMANDS	FORM	DESCRIPTION
STATus:DEVice:CONDition?	query only NR1	Read and clear the Device Status condition register[1].
STATus:DEVice:ENABle <num>	NR1	Set and query bits in the Device Status enable register.[2]
STATus:DEVice[:EVENt]?	query only NR1	Read and clear the Device Status event register.[1]
STATus:DEVice:NTRansition <num>	NR1	Set and query bits in the Device Status negative transition register.[2]
STATus:DEVice:PTRansition <num>	NR1	Set and query bits in the Device Status positive transition register.[2]
STATus:OPERation:AVERaging :CONDition?	query only NR1	Read the Averaging status condition register.[1]
STATus:OPERation:AVERaging:ENABle <num>	NR1	Set and query bits in the Averaging status enable register.[2]
STATus:OPERation:AVERaging[:EVENt]?	query only NR1	Read and clear the Averaging status event register.[1]
STATus:OPERation:AVERaging :NTRansition <num>	NR1	Set and query bits in the Averaging status negative transition register.[2]
STATus:OPERation:AVERaging :PTRansition <num>	NR1	Set and query bits in the Averaging status positive transition register.[2]
STATus:OPERation:CONDition?	query only NR1	Read the Operational Status condition register.[1]
STATus:OPERation:ENABle <num>	NR1	Set and query bits in the Operational Status enable register.[2]
STATus:OPERation[:EVENt]?	query only NR1	Read and clear the Operational Status event register.[1]

1 Returns the sum of the decimal weights (2^n where n is the bit number) of all bits currently set. For more information on using the status registers refer to Chapter 5, "Using Status Registers."

2 **num** is the sum of the decimal weights of all bits to be set.

STATus (continued)

SUBSYSTEM COMMANDS	FORM	DESCRIPTION
STATus:OPERation:MEASuring :CONDition?	query only NR1	Read the Measuring status condition register.[1]
STATus:OPERation:MEASuring:ENABle <num>	NR1	Set and query bits in the Measuring status enable register.[2]
STATus:OPERation:MEASuring[:EVENt]?	query only NR1	Read and clear the Measuring status event register.[1]
STATus:OPERation:MEASuring :NTRansition <num>	NR1	Set and query bits in the Measuring status negative transition register.[2]
STATus:OPERation:MEASuring :PTRansition <num>	NR1	Set and query bits in the Measuring status positive transition register.[2]
STATus:OPERation:NTRansition <num>	NR1	Set and query bits in the Operational Status negative transition register.[2]
STATus:OPERation:PTRansition <num>	NR1	Set and query bits in the Operational Status positive transition register.[2]
STATus:PRESet	command only	Set bits in most enable and transition registers to their default state.
STATus:QUEStionable:CONDition?	query only NR1	Read and clear the Questionable Status condition register.[1]
STATus:QUEStionable:ENABle <num>	NR1	Set and query bits in the Questionable Status enable register.[2]
STATus:QUEStionable[:EVENt]?	query only NR1	Read and clear the Questionable Status event register.[1]
STATus:QUEStionable:LIMit :CONDition?	query only NR1	Read and clear the Limit Fail condition register.[1]
STATus:QUEStionable:LIMit:ENABle <num>	NR1	Set and query bits in the Limit Fail enable register.[2]

1 Returns the sum of the decimal weights (2^n where n is the bit number) of all bits currently set. For more information on using the status registers refer to Chapter 5, "Using Status Registers."

2 **num** is the sum of the decimal weights of all bits to be set.

STATus (continued)

SUBSYSTEM COMMANDS	FORM	DESCRIPTION
STATus:QUEStionable:LIMit[:EVENt]?	query only NR1	Read and clear the Limit Fail event register.[1]
STATus:QUEStionable:LIMit :NTRansition <num>	NR1	Set and query bits in the Limit Fail negative transition register.[2]
STATus:QUEStionable:LIMit :PTRansition <num>	NR1	Set and query bits in the Limit Fail positive transition register.[2]
STATus:QUEStionable:NTRansition <num>	NR1	Set and query bits in the Questionable Status negative transition register.[2]
STATus:QUEStionable:PTRansition <num>	NR1	Set and query bits in the Questionable Status positive transition register.[2]

1 Returns the sum of the decimal weights (2^n where n is the bit number) of all bits currently set. For more information on using the status registers refer to Chapter 5, "Using Status Registers."

2 **num** is the sum of the decimal weights of all bits to be set.

SYSTem

SUBSYSTEM COMMANDS	FORM	DESCRIPTION
SYSTem:BEEPer[:IMMediate] [<freq>[,<dur>[,<vol>]]][1]	NR3, NR3, NR3	Instructs the analyzer to beep. Arguments are frequency (Hz), duration (seconds), and volume (0 to 1).
SYSTem:BEEPer:VOLume <num>	NR2	Set the volume of the beeper — num is a number between 0 for 0% and 1 for 100%.
SYSTem:COMMunicate:GPIB:CONTroller [:STATe] <ON|OFF>[2,3]	NR1	Makes the analyzer the system controller.
SYSTem:COMMunicate:GPIB:ECHO <ON|OFF>[2]	NR1	Turn on/off HP-IB mnemonic echo.
SYSTem:COMMunicate:GPIB:HCOPy :ADDRess <num>	NR1	Set the address of an HP-IB printer or plotter for hardcopy output — num must be an integer between 0 and 30.
SYSTem:COMMunicate:GPIB[:SELF] :ADDRess <num>[4]	NR1	Set the analyzer's HP-IB address — num must be an integer between 0 and 30.
SYSTem:COMMunicate:LAN:EADDress?	query only	Queries the analyzer's ethernet address.
SYSTem:COMMunicate:LAN:IPADdress <string>	STRING	Sets the analyzer's Internet Protocol address.
SYSTem:COMMunicate:LAN:PRINter:HOSTname <string>	STRING	Specifies the IP address of the LAN printer.
SYSTem:COMMunicate:LAN:ROUTe:GATeway <string>	STRING	Sets the IP address for a LAN gateway.
SYSTem:COMMunicate:LAN:ROUTe:SMASk <string>	STRING	Sets the subnet mask.
SYSTem:COMMunicate:LAN:STATe <ON|OFF>[2]	STRING	Turns networking on or off.
SYSTem:COMMunicate:SERial:TRANsmit :BAUD <num>	NR1	Set the baud rate for hardcopy output to a device on the serial port — choose from 1200|2400|4800|9600|19200.
SYSTem:COMMunicate:SERial:TRANsmit :HANDshake <char>	CHAR	Set the handshake for communication to a hardcopy device on the serial port — choose XON or DTR.

1 <**freq**>, <**dur**>, and <**vol**> are optional <**num**> parameters.

2 Binary parameters accept the values of **1** (on) and **0** (off) in addition to **ON** and **OFF**.

3 For use with IBASIC — this command cannot be executed from an external controller.

4 A delay of 5 seconds is required before a command is sent to the new address.

SYSTem (continued)

SUBSYSTEM COMMANDS	FORM	DESCRIPTION
SYSTem:COMMunicate:TTL:USER:FEED <char>	CHAR	Selects the function of the USER TTL IN/OUT port on the rear panel of the analyzer. Choose from DEFault|KEY|SWEep.
SYSTem:DATE <num1>,<num2>,<num3>	NR1,NR1, NR1	Set the year (num1), month (num2) and day (num3) of the real time clock.
SYSTem:ERRor?[1]	query only NR1,STRING	Query the error queue — returns the error number and message.
SYSTem:KEY <char>	command only	Sends key names[3] which execute the same functions as front panel keys.
SYSTem:KEY:MASK?	query only NR1	Query the mask (shift, ctrl, alt) associated with a keypress on an external keyboard.
SYSTem:KEY:QUEue:CLEar	command only	Clears the key queue.
SYSTem:KEY:QUEue:COUNt?	query only NR1	Query the number of key codes in the queue.
SYSTem:KEY:QUEue:MAXimum?	query only NR1	Query the size of the key queue (the maximum number of key codes it can hold).
SYSTem:KEY:QUEue[:STATe] <ON|OFF>[2]	NR1	Turn on/off the key queue.
SYSTem:KEY:TYPE?	query only CHAR	Query the type of key that was pressed — returns NONE, RPG, KEY (front panel key) or ASC (external keyboard).
SYSTem:KEY:USER	command only	Sets the User Request bit of the Standard Event Status Register.
SYSTem:KEY[:VALue]?	query only NR1	Query the key code value for the last key pressed — RPG type returns the knob count, positive for clockwise rotation, KEY type returns the front panel keycode,[3] and ASC type returns the ASCII code number.

1 For more information on errors, refer to Chapter 14, "SCPI Error Messages."

2 Binary parameters accept the values of **1** (on) and **0** (off) in addition to **ON** and **OFF**.

3 A list of the analyzer's front panel keycodes and key names is provided in Chapter 9.

SYSTem (continued)

SUBSYSTEM COMMANDS	FORM	DESCRIPTION
SYSTem:PRESet	command only	Perform a system preset — this is the same as the front panel (PRESET) key.
SYSTem:SET <block>	command only	Send a learn string (obtained using *LRN?) to the analyzer — this command is included in the learn string.
SYSTem:SET:LRN? [<USER>][1]	BLOCK	Query or set the instrument state.
SYSTem:SET:LRNLong? [<USER>][1]	BLOCK	Query or set the instrument state, data, and calibration. Similar to save/recall.
SYSTem:TIME <num1>,<num2>,<num3>	NR1,NR1, NR1	Set the hour (num1), minute (num2) and second (num3) of the real time clock.
SYSTem:VERSion?	query only NR2	Query the SCPI version of the analyzer. See *IDN? to query the firmware revision.

1 Refer to "Automated Measurement Setup and Control" in Chapter 7 of the *User's Guide* for more information on using this command.

TEST

SUBSYSTEM COMMANDS	FORM	DESCRIPTION
TEST:RESult?	query only CHAR	Query the result of the selected adjustment or self-test — the response will be NULL|PASS|FAIL.
TEST:SELect <num>	NR1	Select the adjustment or self-test to execute.
TEST:STATe <char>	CHAR	Select the state of the active adjustment or self-test — choose from RUN|CONTinue|STOP for the command. Query returns NULL|RUN|PAUS|DONE.
TEST:VALue <num>	NR1	Set or query a value for an adjustment or self-test.

TRACe

<table>
<tr><th>SUBSYSTEM COMMANDS</th><th>FORM</th><th>DESCRIPTION</th></tr>
<tr><td>TRACe[:DATA]? <char></td><td>query only
BLOCK or NR3[1]</td><td>Query trace data — choose from
CH<1|2>FDATA formatted data,
CH<1|2>FMEM formatted memory,
CH<1|2>SDATA unformatted data,
CH<1|2>SMEM unformatted memory,
CH<1|2><A|B|R>FWD raw data, or
CH<1|2>SCORR<1|2|3|4> correction data. Note: See Chapter 6, "Trace Data Transfers," for data array details.</td></tr>
<tr><td>TRACe[:DATA] <char>,<data></td><td>command only</td><td>Input trace data — choose from the above list of arrays. The data can be either BLOCK or NR3 type.[1] See Chapter 6 for more information.</td></tr>
<tr><td>TRACe[:DATA] <char1>,<char2></td><td>command only</td><td>Move data from one internal array to another — char1 is the target array (CH<1|2>SMEM) while char2 is the source array (CH<1|2>SDATA). Note that the source and target arrays must be from the same measurement channel.</td></tr>
</table>

1 The parameter type of the data is determined by the format selected — **FORMat REAL** uses BLOCK data, **FORMat ASCii** uses NR3 data separated by commas.

TRIGger

SUBSYSTEM COMMANDS	FORM	DESCRIPTION
TRIGger[:SEQuence]:SOURce <char>	CHAR	Set the source for the sweep trigger signal — choose IMMediate or EXTernal (used in conjunction with SENSe:SWEep:TRIGger:SOURce).

13

SCPI Conformance Information

SCPI Conformance Information

The HP 8711C/12C/13C/14C RF Network Analyzers conform to the 1996.0 version of SCPI.

SCPI Standard Commands

The analyzer implements the following IEEE 488.2 standard commands:

```
*CLS
*ESE
*ESE?
*ESR?
*IDN?
*LRN?
*OPC
*OPC?
*OPT?
*PCB
*PSC
*RST
*SRE
*SRE?
*STB?
*TRG
*TST?
*WAI
```

The analyzer implements the following SCPI 1996.0 standard commands:

```
ABORt

CALCulate[1|2]:DATA?
CALCulate[1|2]:FORMat
CALCulate[1|2]:FORMat?
⊛CALCulate[1|2]:GDAPerture:APERture
⊛CALCulate[1|2]:GDAPerture:SPAN
CALCulate[1|2]:LIMit:STATe
CALCulate[1|2]:LIMit:STATe?
CALCulate[1|2]:MATH[:EXPRession]
CALCulate[1|2]:MATH[:EXPRession]?

CALibration:ZERO:AUTO
CALibration:ZERO:AUTO?

DISPlay:CMAP:COLor[1|2| ... 16]:HSL
DISPlay:CMAP:COLor[1|2| ... 16]:HSL?
```

```
DISPlay:CMAP:COLor[1|2| ... 16]:RGB
DISPlay:CMAP:COLor[1|2| ... 16]:RGB?
DISPlay:CMAP:DEFault
DISPlay:MENU[1|2]:KEY[1|2| ... 7]?
DISPlay:WINDow[1|2|10]:GEOMetry:LLEFT?
DISPlay:WINDow[1|2|10]:GEOMetry:SIZE?
DISPlay:WINDow[1|2|10]:GEOMetry:URIGHT?
DISPlay:WINDow[1|2|10]:GRAPhics:CLEar
DISPlay:WINDow[1|2|10]:GRAPhics:COLor
DISPlay:WINDow[1|2|10]:GRAPhics:COLor?
DISPlay:WINDow[1|2|10]:GRAPhics[:DRAW]
DISPlay:WINDow[1|2|10]:GRAPhics:LABel
DISPlay:WINDow[1|2|10]:GRAPhics:MOVE
DISPlay:WINDow[1|2|10]:GRAPhics:MOVE?
DISPlay:WINDow[1|2|10]:GRAPhics:STATe?
DISPlay:WINDow[1|2]:TRACe:GRATicule:GRID[:STATe]
DISPlay:WINDow[1|2]:TRACe:GRATicule:GRID[:STATe]?
DISPlay:WINDow[1|2]:TRACe[1|2][:STATe]
DISPlay:WINDow[1|2]:TRACe[1|2][:STATe]?
DISPlay:WINDow[1|2]:TRACe:Y[:SCALe]:AUTO
DISPlay:WINDow[1|2]:TRACe:Y[:SCALe]:PDIVision
DISPlay:WINDow[1|2]:TRACe:Y[:SCALe]:PDIVision?
DISPlay:WINDow[1|2]:TRACe:Y[:SCALe]:RLEVel
DISPlay:WINDow[1|2]:TRACe:Y[:SCALe]:RLEVel?
DISPlay:WINDow[1|2]:TRACe:Y[:SCALe]:RPOSition
DISPlay:WINDow[1|2]:TRACe:Y[:SCALe]:RPOSition?

FORMat:BORDer
FORMat:BORDer?
FORMat[:DATA]
FORMat[:DATA]?
```

```
HCOPy:ABORt
HCOPy:DEVice[1|2|3]:COLor
HCOPy:DEVice[1|2|3]:COLor?
HCOPy:DEVice[1|2|3]:LANGuage
HCOPy:DEVice[1|2|3]:LANGuage?
HCOPy:DEVice[1|2|3]:MODE
HCOPy:DEVice[1|2|3]:MODE?
HCOPy:DEVice[1|2|3]:RESolution
HCOPy:DEVice[1|2|3]:RESolution?
HCOPy[:IMMediate]
HCOPy:ITEM:ANNotation:STATe
HCOPy:ITEM:ANNotation:STATe?
HCOPy:ITEM[1|2|3]:FFEed:STATe
HCOPy:ITEM[1|2|3]:FFEed:STATe?

INITiate[1|2]:CONTinuous
INITiate[1|2]:CONTinuous?
INITiate[1|2][:IMMediate]

MMEMory:CATalog?
MMEMory:CDIRectory
MMEMory:CDIRectory?
MMEMory:COPY
MMEMory:DELete
MMEMory:FILE:INFO?
MMEMory:INITialize
MMEMory:LOAD:STATe
MMEMory:MOVE
MMEMory:MSIS
MMEMory:MSIS?
MMEMory:STORe:STATe
MMEMory:STORe:TRACe
MMEMory:TRANsfer:BDAT
MMEMory:TRANsfer[:HFS]

OUTPut[:STATe]
OUTPut[:STATe]?
```

```
PROGram:CATalog?
PROGram[:SELected]:DEFine
PROGram[:SELected]:DEFine?
PROGram[:SELected]:DELete:ALL
PROGram[:SELected]:DELete[:SELected]
PROGram[:SELected]:EXECute
PROGram[:SELected]:MALLocate
PROGram[:SELected]:MALLocate?
PROGram[:SELected]:NAME
PROGram[:SELected]:NAME?
PROGram[:SELected]:NUMBer
PROGram[:SELected]:NUMBer?
PROGram[:SELected]:STATe
PROGram[:SELected]:STATe?
PROGram[:SELected]:STRing
PROGram[:SELected]:STRing?
PROGram[:SELected]:WAIT
PROGram[:SELected]:WAIT?

SENSe[1|2]:AVERage:COUNt
SENSe[1|2]:AVERage:COUNt?
SENSe[1|2]:AVERage[:STATe]
SENSe[1|2]:AVERage[:STATe]?
SENSe[1|2]:BWIDth[:RESolution]
SENSe[1|2]:BWIDth[:RESolution]?
SENSe[1|2]:CORRection:COLLect[:ACQuire]
SENSe[1|2]:CORRection:COLLect:METHod
SENSe[1|2]:CORRection:COLLect:SAVE
SENSe[1|2]:CORRection:CSET[:SELect]
SENSe[1|2]:CORRection:CSET[:SELect]?
⊛SENSe[1|2]:CORRection:EDELay:TIME
⊛SENSe[1|2]:CORRection:IMPedance:INPut:MAGNitude
⊛SENSe[1|2]:CORRection:OFFSet:PHASe
⊛SENSe[1|2]:CORRection:RVELocity:COAX
SENSe[1|2]:CORRection[:STATe]
SENSe[1|2]:CORRection[:STATe]?
SENSe[1|2]:DETector[:FUNCTION]
SENSe[1|2]:FREQuency:CENTer
SENSe[1|2]:FREQuency:CENTer?
SENSe[1|2]:FREQuency:SPAN
```

```
SENSe[1|2]:FREQuency:SPAN?
SENSe[1|2]:FREQuency:STARt
SENSe[1|2]:FREQuency:STARt?
SENSe[1|2]:FREQuency:STOP
SENSe[1|2]:FREQuency:STOP?
SENSe[1|2]:FUNCtion
SENSe[1|2]:FUNCtion?
SENSe:ROSCillator:SOURce
SENSe:ROSCillator:SOURce?
SENSe[1|2]:SWEep:POINts
SENSe[1|2]:SWEep:POINts?
SENSe[1|2]:SWEep:TIME
SENSe[1|2]:SWEep:TIME?
SENSe[1|2]:SWEep:TIME:AUTO
SENSe[1|2]:SWEep:TIME:AUTO?

SOURce[1|2]:POWer[:LEVel][:IMMediate][:AMPLitude]
SOURce[1|2]:POWer[:LEVel][:IMMediate][:AMPLitude]?
SOURce[1|2]:POWer:RANGe
SOURce[1|2]:POWer:STARt
SOURce[1|2]:POWer:STOP

STATus:OPERation:CONDition?
STATus:OPERation:ENABle
STATus:OPERation:ENABle?
STATus:OPERation[:EVENt]?
STATus:OPERation:NTRansition
STATus:OPERation:NTRansition?
STATus:OPERation:PTRansition
STATus:OPERation:PTRansition?
STATus:QUEStionable:CONDition?
STATus:QUEStionable:ENABle
STATus:QUEStionable:ENABle?
STATus:QUEStionable[:EVENt]?
STATus:QUEStionable:NTRansition
STATus:QUEStionable:NTRansition?
STATus:QUEStionable:PTRansition
STATus:QUEStionable:PTRansition?

SYSTem:BEEPer[:IMMediate]?
SYSTem:BEEPer:VOLume
SYSTem:BEEPer:VOLume?
```

```
SYSTem:COMMunicate:GPIB[:SELF]:ADDRess
SYSTem:COMMunicate:GPIB[:SELF]:ADDRess?
SYSTem:COMMunicate:SERial:TRANsmit:BAUD
SYSTem:COMMunicate:SERial:TRANsmit:BAUD?
SYSTem:DATE
SYSTem:DATE?
SYSTem:ERRor?
SYSTem:KEY[:VALue]?
SYSTem:PRESet
SYSTem:SET
SYSTem:SET:LRN?
SYSTem:TIME
SYSTem:TIME?
SYSTem:VERSion?

TRACe[:DATA]
TRACe[:DATA]?

TRIGger[:SEQuence]:SOURce
TRIGger[:SEQuence]:SOURce?
```

Instrument Specific Commands

The following are instrument specific commands implemented by the HP 8711C/12C/13C/14C RF Network Analyzers which are not part of the present SCPI 1996.0 definition.

```
CALCulate[1|2]:FORMat:UNIT:MLIN
CALCulate[1|2]:FORMat:UNIT:MLIN?
CALCulate[1|2]:FORMat:UNIT:MLOG
CALCulate[1|2]:FORMat:UNIT:MLOG?
CALCulate[1|2]:LIMit:DISPlay
CALCulate[1|2]:LIMit:DISPlay?
CALCulate[1|2]:LIMit:MARKer:FLATness:MAXimum
CALCulate[1|2]:LIMit:MARKer:FLATness:MINimum
CALCulate[1|2]:LIMit:MARKer:FLATness[:STATe]
CALCulate[1|2]:LIMit:MARKer:FREQuency:MAXimum
CALCulate[1|2]:LIMit:MARKer:FREQuency:MINimum
CALCulate[1|2]:LIMit:MARKer:FREQuency[:STATe]
CALCulate[1|2]:LIMit:MARKer:STATistic:MEAN:MAXimum
CALCulate[1|2]:LIMit:MARKer:STATistic:MEAN:MINimum
CALCulate[1|2]:LIMit:MARKer:STATistic:MEAN[:STATe]
CALCulate[1|2]:LIMit:MARKer:STATistic:PEAK:MAXimum
CALCulate[1|2]:LIMit:MARKer:STATistic:PEAK:MINimum
CALCulate[1|2]:LIMit:MARKer:STATistic:PEAK[:STATe]
CALCulate[1|2]:LIMit:MARKer:TILT:MAXimum
CALCulate[1|2]:LIMit:MARKer:TILT:MINimum
CALCulate[1|2]:LIMit:MARKer:TILT[:STATe]
CALCulate[1|2]:LIMit:SEGMent[1|2| ... 12]:AMPLitude:STARt
CALCulate[1|2]:LIMit:SEGMent[1|2| ... 12]:AMPLitude:STARt?
CALCulate[1|2]:LIMit:SEGMent[1|2| ... 12]:AMPLitude:STOP
CALCulate[1|2]:LIMit:SEGMent[1|2| ... 12]:AMPLitude:STOP?
CALCulate[1|2]:LIMit:SEGMent:AOFF
CALCulate[1|2]:LIMit:SEGMent[1|2| ... 12]:FREQuency:STARt
CALCulate[1|2]:LIMit:SEGMent[1|2| ... 12]:FREQuency:STARt?
CALCulate[1|2]:LIMit:SEGMent[1|2| ... 12]:FREQuency:STOP
CALCulate[1|2]:LIMit:SEGMent[1|2| ... 12]:FREQuency:STOP?
CALCulate[1|2]:LIMit:SEGMent[1|2| ... 12]:POWer:STOP
CALCulate[1|2]:LIMit:SEGMent[1|2| ... 12]:POWer:STOP?
CALCulate[1|2]:LIMit:SEGMent[1|2| ... 12]:STATe
CALCulate[1|2]:LIMit:SEGMent[1|2| ... 12]:STATe?
```

```
CALCulate[1|2]:LIMit:SEGMent[1|2| ... 12]:TYPE
CALCulate[1|2]:LIMit:SEGMent[1|2| ... 12]:TYPE?
CALCulate[1|2]:MARKer:AOFF
CALCulate[1|2]:MARKer:BWIDth
CALCulate[1|2]:MARKer:BWIDth?
CALCulate[1|2]:MARKer:FUNCtion:RESult?
CALCulate[1|2]:MARKer:FUNCtion[:SELect]
CALCulate[1|2]:MARKer:FUNCtion[:SELect]?
CALCulate[1|2]:MARKer:FUNCtion:TRACking
CALCulate[1|2]:MARKer:FUNCtion:TRACking?
⊛CALCulate[1|2]:MARKer[1|2| ... 8]:GDELay?
CALCulate[1|2]:MARKer[1|2| ... 8]:MAXimum
CALCulate[1|2]:MARKer[1|2| ... 8]:MAXimum:LEFT
CALCulate[1|2]:MARKer[1|2| ... 8]:MAXimum:RIGHt
CALCulate[1|2]:MARKer[1|2| ... 8]:MINimum
CALCulate[1|2]:MARKer[1|2| ... 8]:MINimum:LEFT
CALCulate[1|2]:MARKer[1|2| ... 8]:MINimum:RIGHt
CALCulate[1|2]:MARKer:MODE
CALCulate[1|2]:MARKer:MODE?
CALCulate[1|2]:MARKer:NOTCh
CALCulate[1|2]:MARKer[1|2| ... 8]:POINt
CALCulate[1|2]:MARKer[1|2| ... 8]:POINt?
CALCulate[1|2]:MARKer:REFerence:X?
CALCulate[1|2]:MARKer:REFerence:Y?
CALCulate[1|2]:MARKer[1|2| ... 8][:STATe]
CALCulate[1|2]:MARKer[1|2| ... 8][:STATe]?
CALCulate[1|2]:MARKer[1|2| ... 8]:TARGet
CALCulate[1|2]:MARKer[1|2| ... 8]:TARGet?
CALCulate[1|2]:MARKer[1|2| ... 8]:X
CALCulate[1|2]:MARKer[1|2| ... 8]:X?
CALCulate[1|2]:MARKer[1|2| ... 8]:X:ABS
CALCulate[1|2]:MARKer[1|2| ... 8]:Y?
CALCulate[1|2]:MARKer[1|2| ... 8]:Y:INDuctance?
CALCulate[1|2]:MARKer[1|2| ... 8]:Y:MAGNitude?
⊛ CALCulate[1|2]:MARKer[1|2| ... 8]:Y:PHASe?
⊛CALCulate[1|2]:MARKer[1|2| ... 8]:Y:REACtance?
⊛CALCulate[1|2]:MARKer[1|2| ... 8]:Y:RESistance?

CALibration:SELF
CALibration:SELF:TIMER
CALibration:SELF:ALL
```

```
CONFigure
CONFigure?
CONTrol[1|2]:MULTiport:STATE
DIAGnostic:CCONstants:INSTalled?
DIAGnostic:CCONstants:LOAD
DIAGnostic:CCONstants:STORe:DISK
DIAGnostic:CCONstants:STORe:EEPRom
DIAGnostic:COMMunicate:LAN:PING:IMM (Option 1F7 only)
DIAGnostic:COMMunicate:LAN:PING:IPADress (Option 1F7 only)
DIAGnostic:COMMunicate:LAN:SEND (Option 1F7 only)
DIAGnostic:MDISplay[1|2]:CORRection C_DIRECT
DIAGnostic:MDISplay[1|2]:CORRection C_ISOLATION
DIAGnostic:MDISplay[1|2]:CORRection C_LDMATCH
DIAGnostic:MDISplay[1|2]:CORRection C_RTRACKING
DIAGnostic:MDISplay[1|2]:CORRection C_SRCMATCH
DIAGnostic:MDISplay[1|2]:CORRection C_TTRACKING
DIAGnostic:MDISplay[1|2]:CORRection I_DIRECTivity
DIAGnostic:MDISplay[1|2]:CORRection I_RESPONSE
DIAGnostic:MDISplay[1|2]:CORRection I_SRCMATCH
DIAGnostic:MDISplay[1|2]:CORRection I_TRACKING
DIAGnostic:MDISplay[1|2]:CORRection M_DIRECTivity
DIAGnostic:MDISplay[1|2]:CORRection M_RESPONSE
DIAGnostic:MDISplay[1|2]:CORRection M_SRCMATCH
DIAGnostic:MDISplay[1|2]:CORRection M_TRACKING
DIAGnostic:MDISplay[1|2]:CORRection M_XSCALAR
DIAGnostic:MDISplay[1|2]:REST
DIAGnostic:DITHer
DIAGnostic:DITHer?
DIAGnostic:SNUMber
DIAGnostic:SNUMber?
DIAGnostic:SPUR:AVOid
DIAGnostic:SPUR:AVOid?
```

```
DISPlay:ANNotation:CHANnel[1|2][:STATe]
DISPlay:ANNotation:CHANnel[1|2]:USER:LABel[:DATA]
DISPlay:ANNotation:CHANnel[1|2]:USER[:STATe]
DISPlay:ANNotation:CLOCk:DATE:FORMat
DISPlay:ANNotation:CLOCk:DATE:FORMat?
DISPlay:ANNotation:CLOCk:DATE:MODE
DISPlay:ANNotation:CLOCk:DATE:MODE?
DISPlay:ANNotation:CLOCk:MODE
DISPlay:ANNotation:CLOCk:MODE?
DISPlay:ANNotation:CLOCk:SEConds[:STATe]
DISPlay:ANNotation:CLOCk:SEConds[:STATe]?
DISPlay:ANNotation:FREQuency[1|2]:MODE
DISPlay:ANNotation:FREQuency[1|2]:MODE?
DISPlay:ANNotation:FREQuency:RESolution
DISPlay:ANNotation:FREQuency:RESolution?
DISPlay:ANNotation:FREQuency[1|2][:STATe]
DISPlay:ANNotation:FREQuency[1|2]:USER:LABel[:DATA]
DISPlay:ANNotation:FREQuency[1|2]:USER:STARt
DISPlay:ANNotation:FREQuency[1|2]:USER[:STATe]
DISPlay:ANNotation:FREQuency[1|2]:USER:STOP
DISPlay:ANNotation:FREQuency[1|2]:USER:SUFFIX
DISPlay:ANNotation:LIMit:ICON[1|2]:FLAG
DISPlay:ANNotation:LIMit:ICON[1|2]:POS:X
DISPlay:ANNotation:LIMit:ICON[1|2]:POS:Y
DISPlay:ANNotation:LIMit:ICON[1|2]:TEXT
DISPlay:ANNotation:LIMit:ICON[1|2]:STATe
DISPlay:ANNotation:MARKer[1|2]:NUMBers[:STATe]
DISPlay:ANNotation:MARKer[1|2][:STATe]
DISPlay:ANNotation:MARKer[1|2][:STATe]?
DISPlay:ANNotation:MESSage:AOFF
DISPlay:ANNotation:MESSage:CLEar
DISPlay:ANNotation:MESSage[:DATA]?
DISPlay:ANNotation:MESSage:STATe
DISPlay:ANNotation:MESSage:STATe?
DISPlay:ANNotation:TITLe[1|2]:DATA
DISPlay:ANNotation:TITLe[1|2]:DATA?
DISPlay:ANNotation:TITLe[:STATe]
DISPlay:ANNotation:TITLe[:STATe]?
DISPlay:ANNotation:YAXis:MODE
DISPlay:ANNotation:YAXis:MODE?
```

```
DISPlay:ANNotation:YAXis[:STATe]
DISPlay:ANNotation:YAXis[:STATe]?
DISPlay:CMAP:COLor[1|2| ... 16]:GREYscale
DISPlay:CMAP:SCHeme
DISPlay:FORMat
DISPlay:FORMat?
DISPlay:FORMat:EXPAND
DISPlay:MENU:RECall:FAST[:STATe]
DISPlay:PROGram[:MODE]
DISPlay:PROGram[:MODE]?
DISPlay:WINDow:GRAPhics:BUFFer[:STATe]
DISPlay:WINDow:GRAPhics:BUFFer[:STATe]?
DISPlay:WINDow[1|2|10]:GRAPhics:CIRCle
DISPlay:WINDow[1|2|10]:GRAPhics:LABel:FONT
DISPlay:WINDow[1|2|10]:GRAPhics:LABel:FONT?
DISPlay:WINDow[1|2|10]:GRAPhics:RECTangle
DISPlay:WINDow[1|2|10]:TRACe[1|2]:Y:TRACk

HCOPy:DEVice:PAGE:MARGin:LEFT
HCOPy:DEVice:PAGE:MARGin:TOP
HCOPy:DEVice:PAGE:ORIentation
HCOPy:DEVice:PAGE:WIDTh
HCOPy:DEVice:PORT
HCOPy:DEVice:PORT?
HCOPy:ITEM:GRATicule:STATe
HCOPy:ITEM:GRATicule:STATe?
HCOPy:ITEM:MARKer:STATe
HCOPy:ITEM:MARKer:STATe?
HCOPy:ITEM:TITLe:STATe
HCOPy:ITEM:TITLe:STATe?
HCOPy:ITEM:TRACe:STATe
HCOPy:ITEM:TRACe:STATe?
HCOPy:PAGE:MARGin:LEFT
HCOPy:PAGE:MARGin:LEFT?
HCOPy:PAGE:MARGin:TOP
HCOPy:PAGE:MARGin:TOP?
HCOPy:PAGE:ORIentation
HCOPy:PAGE:ORIentation?
HCOPy:PAGE:WIDTh
HCOPy:PAGE:WIDTh?
```

INPut:GAIN:AUTO
INPut:GAIN:SETTing

MMEMory:MDIRectory
MMEMory:RDIRectory
MMEMory:STORe:STATe:CORRection
MMEMory:STORe:STATe:CORRection?
MMEMory:STORe:STATe:ISTate
MMEMory:STORe:STATe:ISTate?
MMEMory:STORe:STATe:TRACe
MMEMory:STORe:STATe:TRACe?
MMEMory:STORe:STATe:TSCAL
MMEMory:STORe:TRACe
MMEMory:STORe:TRACe:FORMat
MMEMory:TRANsfer:BDAT
MMEMory:TRANsfer[:HFS]

POWer\[1|2]:MODE

ROUTe[1|2]:REFLection:PATH:DEFine:PORT
ROUTe[1|2]:TRANsmission:PATH:DEFine:PORT

SENSe[1|2]:AVERage:CLEar
SENSe[1|2]:CORRection:CAPacitance:CONNector (Option 100 only)
SENSe[1|2]:CORRection:CAPacitance:CONNector? (Option 100 only)
SENSe[1|2]:CORRection:COLLect:ABORt
SENSe[1|2]:CORRection:COLLect:CKIT[:SELect]
SENSe[1|2]:CORRection:COLLect:CKIT[:SELect]?
SENSe[1|2]:CORRection:COLLect:ISTate[:AUTO]
SENSe[1|2]:CORRection:COLLect:ISTate[:AUTO]?
SENSe[1|2]:CORRection:COLLect:PORTS
SENSe[1|2]:CORRection:COLLect:MP:OPEN
SENSe[1|2]:CORRection:COLLect:MP:SHORT
SENSe[12]:CORRection:COLLect:MP:LOAD
SENSe[1|2]:CORRection:COLLect:MP:THRU
SENSe[1|2]:CORRection:COLLect:VERify:TRANsmission
SENSe[1|2]:CORRection:COLLect:VERify:REFLection
⊛SENSe[1|2]:CORRection:EXTension[:STATe]
⊛SENSe[1|2]:CORRection:EXTension:REFLection[:TIME]
⊛SENSe[1|2]:CORRection:EXTension:TRANsmission[:TIME]
SENSe[1|2]:CORRection:IMPedance:INPut:MAGNitude:SELect
SENSe[1|2]:CORRection:LENGth:COAX (Option 100 only)
SENSe[1|2]:CORRection:LENGth:COAX? (Option 100 only)

SENSe[1|2]:CORRection:LENGth:CONNector (Option 100 only)
SENSe[1|2]:CORRection:LENGth:CONNector? (Option 100 only)
SENSe[1|2]:CORRection:LOSS:COAX (Option 100 only)
SENSe[1|2]:CORRection:LOSS:COAX? (Option 100 only)
SENSe[1|2]:CORRection:MODel:CONNector[:IMMediate] (Option 100 only)
SENSe[1|2]:CORRection:PEAK:COAX (Option 100 only)
SENSe[1|2]:CORRection:PEAK:COAX? (Option 100 only)
SENSe[1|2]:CORRection:RVELocity[:IMMediate] (Option 100 only)
SENSe[1|2]:CORRection: TESTSET
SENSe[1|2]:CORRection:THReshold:COAX (Option 100 only)
SENSe[1|2]:CORRection:THReshold:COAX? (Option 100 only)
SENSe:COUPle
SENSe:COUPle?
SENSe[1|2]:DETector[:FUNCtion]

SENSe[1|2]:DETector[:FUNCtion]?
SENSe:DISTance:STARt (Option 100 only)
SENSe:DISTance:STARt? (Option 100 only)
SENSe:DISTance:STOP (Option 100 only)
SENSe:DISTance:STOP? (Option 100 only)
SENSe:DISTance:UNITs (Option 100 only)
SENSe:DISTance:UNITs? (Option 100 only)
SENSe:FREQuency:MODE (Option 100 only)
SENSe:FREQuency:MODE? (Option 100 only)
SENSe:FREQuency:SPAN:MAXimum? (Option 100 only)
SENSe:FREQuency:SPAN:MAXimum (Option 100 only)
SENSe:FREQuency:ZSTop (Option 100 only)
SENSe:FREQuency:ZSTop? (Option 100 only)
SENSe:FUNCtion:SRL:IMPedance (Option 100 only)
SENSe:FUNCtion:SRL:IMPedance? (Option 100 only)
SENSe:FUNCtion:SRL:MODE (Option 100 only)
SENSe:FUNCtion:SRL:MODE? (Option 100 only)
SENSe:FUNCtion:SRL:SCAN[:IMMediate] (Option 100 only)
SENSe[1|2]:STATe
SENSe[1|2]:STATe?
SENSe:SWEep:TRIGger:SOURce
SENSe:SWEep:TRIGger:SOURce?
SENSe:WINDow[:TYPE] (Option 100 only)
SENSe:WINDow[:TYPE]? (Option 100 only)

STATus:DEVice:CONDition?
STATus:DEVice:ENABle
STATus:DEVice:ENABle?
STATus:DEVice[:EVENt]?
STATus:DEVice:NTRansition
STATus:DEVice:NTRansition?
STATus:DEVice:PTRansition
STATus:DEVice:PTRansition?
STATus:OPERation:AVERaging:CONDition?
STATus:OPERation:AVERaging:ENABle
STATus:OPERation:AVERaging:ENABle?
STATus:OPERation:AVERaging[:EVENt]?
STATus:OPERation:AVERaging:NTRansition
STATus:OPERation:AVERaging:NTRansition?
STATus:OPERation:AVERaging:PTRansition
STATus:OPERation:AVERaging:PTRansition?

STATus:OPERation:MEASuring:CONDition?
STATus:OPERation:MEASuring:ENABle
STATus:OPERation:MEASuring:ENABle?
STATus:OPERation:MEASuring[:EVENt]?
STATus:OPERation:MEASuring:NTRansition
STATus:OPERation:MEASuring:NTRansition?
STATus:OPERation:MEASuring:PTRansition
STATus:OPERation:MEASuring:PTRansition?
STATus:PRESet
STATus:QUEStionable:LIMit:CONDition?
STATus:QUEStionable:LIMit:ENABle
STATus:QUEStionable:LIMit:ENABle?
STATus:QUEStionable:LIMit[:EVENt]?
STATus:QUEStionable:LIMit:NTRansition
STATus:QUEStionable:LIMit:NTRansition?
STATus:QUEStionable:LIMit:PTRansition
STATus:QUEStionable:LIMit:PTRansition?

SYSTem:COMMunicate:GPIB:CONTroller[:STATe]
SYSTem:COMMunicate:GPIB:CONTroller[:STATe]?
SYSTem:COMMunicate:GPIB:ECHO
SYSTem:COMMunicate:GPIB:ECHO?
SYSTem:COMMunicate:GPIB:HCOPy:ADDRess
SYSTem:COMMunicate:GPIB:HCOPy:ADDRess?
SYSTem:COMMunicate:GPIB:MMEMory:ADDRess
SYSTem:COMMunicate:GPIB:MMEMory:ADDRess?
SYSTem:COMMunicate:GPIB:MMEMory:UNIT
SYSTem:COMMunicate:GPIB:MMEMory:UNIT?
SYSTem:COMMunicate:GPIB:MMEMory:VOLume
SYSTem:COMMunicate:GPIB:MMEMory:VOLume?
SYSTem:COMMunicate:GPIB:MMEMory:VOLume?
SYSTem:COMMunicate:LAN:EADDress? (Option 1F7 only)
SYSTem:COMMunicate:LAN:IPADdress (Option 1F7 only)
SYSTem:COMMunicate:LAN:IPADdress? (Option 1F7 only)
SYSTem:COMMunicate:LAN:PRINter:HOSTname (Option 1F7 only)
SYSTem:COMMunicate:LAN:PRINter:HOSTname? (Option 1F7 only)
SYSTem:COMMunicate:LAN:ROUTe:GATeway (Option 1F7 only)
SYSTem:COMMunicate:LAN:ROUTe:GATeway? (Option 1F7 only)
SYSTem:COMMunicate:LAN:ROUTe:SMASk (Option 1F7 only)
SYSTem:COMMunicate:LAN:ROUTe:SMASk? (Option 1F7 only)
SYSTem:COMMunicate:LAN:STATe (Option 1F7 only)

SYSTem:COMMunicate:LAN:STATe? (Option 1F7 only)
SYSTem:COMMunicate:SERial:TRANsmit:HANDshake
SYSTem:COMMunicate:SERial:TRANsmit:HANDshake?
SYSTem:COMMunicate:TTL:USER:FEED
SYSTem:COMMunicate:TTL:USER:FEED?
SYSTem:KEY:MASK?
SYSTem:KEY:QUEue:CLEar
SYSTem:KEY:QUEue:COUNt?
SYSTem:KEY:QUEue:MAXimum?
SYSTem:KEY:QUEue[:STATe]
SYSTem:KEY:QUEue[:STATe]?
SYSTem:KEY:TYPE?
SYSTem:KEY:USER
SYSTem:SET:LRNLong

TEST:RESult?
TEST:SELect
TEST:SELect?
TEST:STATe
TEST:STATe?
TEST:VALue
TEST:VALue?

14

SCPI Error Messages

SCPI Error Messages

This chapter contains the same error message information that can be found in the *SCPI 1994 Volume 2: Command Reference*. There are four sections in this chapter:

- Command Errors
- Execution Errors
- Device-Specific Errors
- Query Errors

NOTE

Your analyzer does not use all of the error messages listed in this chapter.

Command Errors

An error/event number in the range −199 to −100 indicates that an IEEE 488.2 syntax error has been detected by the instrument's parser. The occurrence of any error in this class shall cause the command error bit (bit 5) in the event status register (IEEE 488.2, section 11.5.1) to be set. One of the following events has occurred:

- An IEEE 488.2 syntax error has been detected by the parser. That is, a controller-to-device message was received which is in violation of the IEEE 488.2 standard. Possible violations include a data element which violates the device listening formats or whose type is unacceptable to the device.
- An unrecognized header was received. Unrecognized headers include incorrect device-specific headers and incorrect or unimplemented IEEE 488.2 common commands.
- A Group Execute Trigger (GET) was entered into the input buffer inside of an IEEE 488.2 program message.

Events that generate command errors shall not generate execution errors, device-specific errors, or query errors; see the other error definitions in this chapter.

Table 14-1. SCPI Command Errors

Error Number	Error Description
−100	Command error — This is the generic syntax error for devices that cannot detect more specific errors. This code indicates only that a Command Error has occurred.
−101	Invalid character — A syntactic element contains a character which is invalid for that type; for example, a header containing an ampersand, SETUP&. This error might be used in place of errors −114, −121, −141, and perhaps some others.
−102	Syntax error — An unrecognized command or data type was encountered; for example, a string was received when the device does not accept strings.
−103	Invalid separator — The parser was expecting a separator and encountered an illegal character; for example, the semicolon was omitted after a program message unit, `*EMC 1:CH1:VOLTS 5`.
−104	Data type error — The parser recognized a data element different than one allowed; for example, numeric or string data was expected but block data was encountered.
−105	GET not allowed — A Group Execute Trigger was received within a program message.
−108	Parameter not allowed — More parameters were received than expected for the header; for example, the `*EMC` common command only accepts one parameter, so receiving `*EMC 0,1` is not allowed.
−109	Missing parameter — Fewer parameters were received than required for the header; for example, the `*EMC` common command requires one parameter, so receiving `*EMC` is not allowed.
−110	Command header error — An error was detected in the header. This error message should be used when the device cannot detect the more specific errors described for errors −111 through −119.
−111	Header separator error — A character which is not a legal header separator was encountered while parsing the header; for example, no white space followed the header, thus `*GMC"MACRO"` is an error.
−112	Program mnemonic too long — The header contains more that twelve characters.
−113	Undefined header — The header is syntactically correct, but it is undefined for this specific device; for example, `*XYZ` is not defined for any device.
−114	Header suffix out of range — The value of a numeric suffix attached to a program mnemonic makes the header invalid.
−120	Numeric data error — This error, as well as errors −121 through −129, are generated when parsing a data element which appears to be numeric, including the nondecimal numeric types. This particular error message should be used if the device cannot detect a more specific error.
−121	Invalid character in number — An invalid character for the data type being parsed was encountered; for example, an alpha in a decimal numeric or a "9" in octal data.
−123	Exponent too large — The magnitude of the exponent was larger than 32000.

Table 14-1. SCPI Command Errors (continued)

Error Number	Error Description
−124	Too many digits — The mantissa of a decimal numeric data element contained more than 255 digits excluding leading zeros.
−128	Numeric data not allowed — A legal numeric data element was received, but the device does not accept one in this position for the header.
−130	Suffix error — This error, as well as errors −131 through −139, are generated when parsing a suffix. This particular error message should be used if the device cannot detect a more specific error.
−131	Invalid suffix — The suffix does not follow the correct syntax, or the suffix is inappropriate for this device.
−134	Suffix too long — The suffix contained more than 12 characters.
−138	Suffix not allowed — A suffix was encountered after a numeric element which does not allow suffixes.
−140	Character data error — This error, as well as errors −141 through −149, are generated when parsing a character data element. This particular error message should be used if the device cannot detect a more specific error.
−141	Invalid character data — Either the character data element contains an invalid character or the particular element received is not valid for the header.
−144	Character data too long — The character data element contains more than twelve characters.
−148	Character data not allowed — A legal character data element was encountered where prohibited by the device.
−150	String data error — This error, as well as errors −151 through −159, are generated when parsing a string data element. This particular error message should be used if the device cannot detect a more specific error.
−151	Invalid string data — A string data element was expected, but was invalid for some reason. For example, an **END** message was received before the terminal quote character.
−158	String data not allowed — A string data element was encountered but was not allowed by the device at this point in parsing.
−160	Block data error — This error, as well as errors −161 through −169, are generated when parsing a block data element. This particular error message should be used if the device cannot detect a more specific error.
−161	Invalid block data — A block data element was expected, but was invalid for some reason. For example, an **END** message was received before the length was satisfied.
−168	Block data not allowed — A legal block data element was encountered but was not allowed by the device at this point in parsing.
−170	Expression error — This error, as well as errors −171 through −179, are generated when parsing an expression data element. This particular error message should be used if the device cannot detect a more specific error.

Table 14-1. SCPI Command Errors (continued)

Error Number	Error Description
−171	Invalid expression — The expression data element was invalid (for example, unmatched parentheses or an illegal character).
−178	Expression data not allowed — A legal expression data was encountered but was not allowed by the device at this point in parsing.
−180	Macro error — This error, as well as errors −181 through −189, are generated when defining or executing a macro. This particular error message should be used if the device cannot detect a more specific error.
−181	Invalid outside macro definition — Indicates that a macro parameter placeholder ($<number>) was encountered outside of a macro definition.
−183	Invalid inside macro definition — Indicates that the program message unit sequence, sent with a `*DDT` or `*DMC` command, is syntactically invalid.
−184	Macro parameter error — Indicates that a command inside the macro definition had the wrong number or type of parameters.

Execution Errors

An error/event number in the range −299 to −200 indicates that an error has been detected by the instrument's execution control block. The occurrence of any error in this class shall cause the execution error bit (bit 4) in the event status register to be set. One of the following events has occurred:

- A program data element following a header was evaluated by the device as outside of its legal input range or is otherwise inconsistent with the device's capabilities.
- A valid program message could not be properly executed due to some device condition.

Execution errors shall be reported by the device after rounding and expression evaluation operations have taken place. Rounding a numeric data element, for example, shall not be reported as an execution error. Events that generate execution errors shall not generate Command Errors, device-specific errors, or Query Errors; see the other error definitions in this section.

Table 14-2. SCPI Execution Errors

Error Number	Error Description
−200	Execution error — This is the generic syntax error for devices that cannot detect more specific errors. This code indicates only that an Execution Error has occurred.
−201	Invalid while in local — Indicates that a command is not executable while the device is in local due to a hard local control; for example, a device with a rotary switch receives a message which would change the switches state, but the device is in local so the message can not be executed.
−202	Settings lost due to rtl — Indicates that a setting associated with a hard local control was lost when the device changed to LOCS from REMS or to LWLS from RWLS.
−203	Command protected — Indicates that a legal password-protected program command or query could not be executed because the command was disabled.
−210	Trigger error
−211	Trigger ignored — Indicates that a **GET**, ***TRG**, or triggering signal was received and recognized by the device but was ignored because of device timing considerations; for example, the device was not ready to respond.[1]
−212	Arm ignored — Indicates that an arming signal was received and recognized by the device but was ignored.
−213	Init ignored — Indicates that a request for a measurement initiation was ignored as another measurement was already in progress.
−214	Trigger deadlock — Indicates that the trigger source for the initiation of a measurement is set to **GET** and subsequent measurement query is received. The measurement cannot be started until a **GET** is received, but the **GET** would cause an **INTERRUPTED** error.
−215s	Arm deadlock — Indicates that the arm source for the initiation of a measurement is set to **GET** and subsequent measurement query is received. The measurement cannot be started until a **GET** is received, but the **GET** would cause an **INTERRUPTED** error.
−220	Parameter error — Indicates that a program data element related error occurred. This error message should be used when the device cannot detect the more specific errors −221 through −229.
−221	Settings conflict — Indicates that a legal program data element was parsed but could not be executed due to the current device state.
−222	Data out of range — Indicates that a legal program data element was parsed but could not be executed because the interpreted value was outside the legal range as defined by the device.
−223	Too much data — Indicates that a legal program data element of block, expression, or string type was received that contained more data than the device could handle due to memory or related device-specific requirements.
−224	Illegal parameter value — Used where an exact value, from a list of possible values, was expected.

1 A DTO device always ignores GET and treats *TRG as a Command Error.

Table 14-2. SCPI Execution Errors (continued)

Error Number	Error Description
−225	Out of memory — The device has insufficient memory to perform the requested operation.
−226	Lists not same length — Attempted to use LIST structure having individual LIST's of unequal lengths.
−230	Data corrupt or stale — Possibly invalid data; new reading started but not completed since last access.
−231	Data questionable — Indicates that measurement accuracy is suspect.
−232	Invalid format — Indicates that a legal program data element was parsed but could not be executed because the data format or structure is inappropriate, such as when loading memory tables or when sending a SYSTem:SET parameter from an unknown instrument.
−233	Invalid version — Indicates that a legal program data element was parsed but could not be executed because the version of the data is incorrect to the device. This particular error should be used when file or block data formats are recognized by the instrument but cannot be executed for reasons of version incompatibility. For example, an unsupported file version, or an unsupported instrument version.
−240	Hardware error — Indicates that a legal program command or query could not be executed because of a hardware problem in the device. Definition of what constitutes a hardware problem is completely device-specific. This error message should be used when the device cannot detect the more specific errors described for errors −241 through −249.
−241	Hardware missing — Indicates that a legal program command or query could not be executed because of missing device hardware; for example, an option was not installed. Definition of what constitutes missing hardware is completely device-specific.
−250	Mass storage error — Indicates that a mass storage error occurred. This error message should be used when the device cannot detect the more specific errors described for errors −251 through −259.
−251	Missing mass storage — Indicates that a legal program command or query could not be executed because of missing mass storage; for example, an option that was not installed. Definition of what constitutes missing mass storage is device-specific.
−252	Missing media — Indicates that a legal program command or query could not be executed because of a missing media; for example, no disk. The definition of what constitutes missing media is device-specific.
−253	Corrupt media — Indicates that a legal program command or query could not be executed because of corrupt media for example, bad disk or wrong format. The definition of what constitutes corrupt media is device-specific.
−254	Media full — Indicates that a legal program command or query could not be executed because the media was full; for example, there is no room on the disk. The definition of what constitutes a full media is device-specific.
−255	Directory full — Indicates that a legal program command or query could not be executed because the media directory was full. The definition of what constitutes a full media directory is device-specific.

Table 14-2. SCPI Execution Errors (continued)

Error Number	Error Description
−256	File name not found — Indicates that a legal program command or query could not be executed because the file name on the device media was not found; for example, an attempt was made to read or copy a nonexistent file. The definition of what constitutes a file not being found is device-specific.
−257	File name error — Indicates that a legal program command or query could not be executed because the file name on the device media was in error; for example, an attempt was made to copy to a duplicate file name. The definition of what constitutes a file name error is device-specific.
−258	Media protected — Indicates that a legal program command or query could not be executed because the media was protected; for example, the write-protect tab on a disk was present. The definition of what constitutes protected media is device-specific.
−260	Expression error — Indicates that an expression program data element related error occurred. This error message should be used when the device cannot detect the more specific errors described for errors −261 through −269.
−261	Math error in expression — Indicates that a syntactically legal expression program data element could not be executed due to a math error; for example, a divide-by-zero was attempted. The definition of math error is device-specific.
−270	Macro error — Indicates that a macro-related execution error occurred. This error message should be used when the device cannot detect the more specific errors −271 through −279.
−271	Macro syntax error — Indicates that a syntactically legal macro program data sequence could not be executed due to a syntax error within the macro definition.
−272	Macro execution error — Indicates that a syntactically legal macro program data sequence could not be executed due to some error in the macro definition.
−273	Illegal macro label — Indicates that the macro label defined in the *DMC command was a legal string syntax, but could not be accepted by the device; for example, the label was too long, the same as a common command header, or contained invalid header syntax.
−274	Macro parameter error — Indicates that the macro definition improperly used a macro parameter placeholder.
−275	Macro definition too long — Indicates that a syntactically legal macro program data sequence could not be executed because the string or block contents were too long for the device to handle.
−276	Macro recursion error — Indicates that a syntactically legal macro program data sequence could not be executed because the device found it to be recursive.

Table 14-2. SCPI Execution Errors (continued)

Error Number	Error Description
−277	Macro redefinition not allowed — Indicates that a syntactically legal macro label in the ***DMC** command could not be executed because the macro label was already defined.
−278	Macro header not found — Indicates that a syntactically legal macro label in the ***GMC?** query could not be executed because the header was not previously defined.
−280	Program error — Indicates that a downloaded program-related execution error occurred. This error message should be used when the device cannot detect the more specific errors −281 through −289. A downloaded program is used to add algorithmic capability to a device. The syntax used in the program and the mechanism for downloading a program is device-specific.
−281	Cannot create program — Indicates that an attempt to create a program was unsuccessful. One reason for failure might include not enough memory.
−282	Illegal program name — The name used to reference a program was invalid; for example, redefining an existing program, deleting a nonexistent program, or in general, referencing a nonexistent program.
−283	Illegal variable name — An attempt was made to reference a nonexistent variable in a program.
−284	Program currently running — Certain operations dealing with programs may be illegal while the program is running; for example, deleting a running program might not be possible.
−285	Program syntax error — Indicates that a syntax error appears in a downloaded program. The syntax used when parsing the downloaded program is device-specific.
−286	Program runtime error
−290	Memory use error — Indicates that a user request has directly or indirectly caused an error related to memory or data_handles (this is not the same as "bad" memory).
−291	Out of memory
−292	Referenced name does not exist
−293	Referenced name already exists
−294	Incompatible type — Indicates that the type or structure of a memory item is inadequate.

Device-Specific Errors

An error/event number in the range −399 to −300 or 1 to 32767 indicates that the instrument has detected an error which is not a command error, a query error, or an execution error. It indicates that some device operations did not properly complete, possibly due to an abnormal hardware or firmware condition. These codes are also used for self-test response errors. The occurrence of any error in this class should cause the device-specific error bit (bit 3) in the event status register to be set.

The meaning of positive error codes is device-dependent and may be enumerated or bit mapped; the error message string for positive error codes is not defined by SCPI and available to the device designer. Note that the string is not optional; if the designer does not wish to implement a string for a particular error, the null string should be sent (for example, 42,""). The occurrence of any error in this class should cause the device-specific error bit (bit 3) in the event status register to be set. Events that generate device-specific errors shall not generate command errors, execution errors, or query errors; see the other error definitions in this section.

Table 14-3. SCPI Device-Specific Errors

Error Number	Error Description
−300	Device-specific error — This is the generic device-dependent error for devices that cannot detect more specific errors. This code indicates only that a Device-Dependent Error has occurred.
−310	System error — Indicates that some error, termed "system error" by the device, has occurred. This code is device-dependent.
−311	Memory error — Indicates that an error was detected in the device's memory. The scope of this error is device-dependent.
−312	PUD memory lost — Indicates that the protected user data saved by the *PUD command has been lost.
−313	Calibration memory lost — Indicates that nonvolatile calibration data used by the *CAL? command has been lost.
−314	Save/recall memory lost — Indicates that the nonvolatile data saved by the *SAV? command has been lost.
−315	Configuration memory lost — Indicates that nonvolatile configuration data saved by the device has been lost. The meaning of this error is device-specific.
−330	Self-test failed
−350	Queue overflow — A specific code entered into the queue in lieu of the code that caused the error. This code indicates that there is no room in the queue and an error occurred but was not recorded.
−360	Communication error — This is the generic communication error for devices that cannot detect the more specific errors −361 through −363.
−361	Parity error in program message — Parity bit not correct when data received, for example, on a serial port.
−362	Framing error in program message — A stop bit was not detected when data was received, for example, on a serial port (for example, a baud rate mismatch).
−363	Input buffer overrun — Software or hardware input buffer on serial port overflows with data caused by improper or nonexistent pacing.

Query Errors

An error/event number in the range −499 to −400 indicates that the output queue control of the instrument has detected a problem with the message exchange protocol. The occurrence of any error in this class shall cause the query error bit (bit 2) in the event status register to be set. These errors correspond to message exchange protocol errors. One of the following is true:

- An attempt is being made to read data from the output queue when no output is either present or pending;
- Data in the output queue has been lost.

Events that generate query errors shall not generate command errors, execution errors, or device-specific errors; see the other error definitions in this section.

Table 14-4. SCPI Query Errors

Error Number	Error Description
−400	Query error — This is the generic query error for devices that cannot detect more specific errors. This code indicates only that a Query Error has occurred.
−410	Query **INTERRUPTED** — Indicates that a condition causing an **INTERRUPTED** Query error occurred; for example, a query followed by **DAB** or **GET** before a response was completely sent.
−420	Query **UNTERMINATED** — Indicates that a condition causing an **UNTERMINATED** Query error occurred; for example, the device was addressed to talk and an incomplete program message was received.
−430	Query **DEADLOCKED** — Indicates that a condition causing a **DEADLOCKED** Query error occurred; for example, both input buffer and output buffer are full and the device cannot continue.
−440	Query **UNTERMINATED** after indefinite response — Indicates that a query was received in the same program message after an query requesting an indefinite response was executed.

Index

Index

R