

# HP 8935 CDMA Cellular/PCS Base Station Test Set

## Programmer's Guide

**Firmware Version: A.01.00 and above**

HP Part Number E6380-90018

**Revision A**

**Printed in U.S.A.**

**November 1997**

**PRELIMINARY**



**This document, part number E6380-90018, is a preliminary manual.**

**Updates will be available six months from the date of this manual. If you would like an updated version when it becomes available, please send us the following information:**

Name \_\_\_\_\_ Job Title \_\_\_\_\_  
Company \_\_\_\_\_ Address \_\_\_\_\_  
Phone \_\_\_\_\_ E-mail \_\_\_\_\_

**This document is on (circle one): CD-ROM Paper**

**E-Mail: karen\_otto@hp.com**

**Fax: HP 8935 Updates, Karen Otto, 509-921-3700**

**Snail Mail: HP 8935 Updates, Karen Otto, Hewlett-Packard Company, 24001E. Mission Avenue, Liberty La, WA, 99019**

© Copyright 1997 Hewlett-Packard Company

---

## **Notice**

Information contained in this document is subject to change without notice.

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

This material may be reproduced by or for the U.S. Government pursuant to the Copyright License under the clause at DFARS 52.227-7013 (APR 1988).

---

## **Manufacturer's Declaration**

This statement is provided to comply with the requirements of the German Sound Emission Directive, from 18 January 1991.

This product has a sound pressure emission (at the operator position) < 70 dB(A).

Sound Pressure  $L_p < 70$  dB(A).

At Operator Position.

Normal Operation.

According to ISO 7779:1988/EN 27779:1991 (Type Test).

**Herstellerbescheinigung**

Diese Information steht im Zusammenhang mit den Anforderungen der Maschinenlärminformationsverordnung vom 18 Januar 1991.

Schalldruckpegel  $L_p < 70$  dB(A).

Am Arbeitsplatz.

Normaler Betrieb.

Nach ISO 7779:1988/EN 27779:1991 (Typprüfung).

---

# Safety

## GENERAL

This product and related documentation must be reviewed for familiarization with safety markings and instructions before operation.

This product has been designed and tested in accordance with IEC Publication 1010, "Safety Requirements for Electronic Measuring Apparatus," and has been supplied in a safe condition. This instruction documentation contains information and warnings which must be followed by the user to ensure safe operation and to maintain the product in a safe condition.

## SAFETY SYMBOLS



Indicates instrument damage can occur if indicated operating limits are exceeded. Refer to the instructions in this guide.



Indicates hazardous voltages.



Indicates earth (ground) terminal

---

### WARNING

**A WARNING note denotes a hazard. It calls attention to a procedure, practice, or the like, which, if not correctly performed or adhered to, could result in personal injury. Do not proceed beyond a WARNING sign until the indicated conditions are fully understood and met.**

---

### CAUTION

A CAUTION note denotes a hazard. It calls attention to an operation procedure, practice, or the like, which, if not correctly performed or adhered to, could result in damage to or destruction of part or all of the product. Do not proceed beyond an CAUTION note until the indicated conditions are fully understood and met.

---

## Safety Considerations for this Instrument

---

WARNING



**This product is a Safety Class I instrument (provided with a protective earthing ground incorporated in the power cord). The mains plug shall only be inserted in a socket outlet provided with a protective earth contact. Any interruption of the protective conductor inside or outside of the product is likely to make the product dangerous. Intentional interruption is prohibited.**



**Do not expose to or operate this instrument in outdoor atmospheric conditions such as direct rain, hail, sleet, snow, icing, sunshine or wind. Operate this instrument only within its specified temperature humidity conditions.**



**This instrument is equipped with internal ground fault circuit interrupter class A.**

- **This device does not protect against electrical shock due to contact with both circuit conductors or a fault in supply wiring to product.**
- **Do not use extension cord to connect this product to power receptacle. Attention-ne pas utiliser de rallonge pour raccorder le detecteur-disjoncteur a la prise de courant.**
- **Replace cordset only with HP 8120 series. Attention - Remplacer uniquement par un cordon amovible numero 8120.**
- **Do not use in wet location. Ne pas utiliser dans un emplacement mouille.**

---

WARNING

**Whenever it is likely that the protection has been impaired, the instrument must be made inoperative and be secured against any unintended operation.**

**If this instrument is to be energized via an autotransformer (for voltage reduction), make sure the common terminal is connected to the earth terminal of the power source.**

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

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

**Servicing instructions are for use by qualified personnel only. To avoid electrical shock, do not perform any servicing unless you are qualified to do so.**

**The opening of covers or removal of parts is likely to expose dangerous voltages. Disconnect the product from all voltage sources while it is being opened.**

**Adjustments described in the manual are performed with power supplied to the instrument while protective covers are removed. Energy available at many points may, if contacted, result in personal injury.**

**The power cord is connected to internal capacitors that may remain live for 5 seconds after disconnecting the plug from its power supply.**

**For Continued protection against fire hazard, replace the line fuse(s) only with 250 V fuse(s) or the same current rating and type (for example, normal blow or time delay). Do not use repaired fuses or short circuited fuseholders. FUSE: T 5.0A**

---

**CAUTION**

Always use the three-prong ac power cord supplied with this product. Failure to ensure adequate earth grounding by not using this cord may cause personal injury and/or product damage.

This product is designed for use in Installation Category II and Pollution Degree 2 per IEC 1010 and IEC 664 respectively. For indoor use only.

This product has autoranging line voltage input, be sure the supply voltage is within the specified range.

Ventilation Requirements: When installing the product in a cabinet, the convection into and out of the product must not be restricted. The ambient temperature (outside the cabinet) must be less than the maximum operating temperature of the product by 4° C for every 100 watts dissipated in the cabinet. If the total power dissipated in the cabinet is greater than 800 watts, then forced convection must be used.

---

To prevent electrical shock, disconnect instrument from mains (line) before cleaning. Use a dry cloth or one slightly dampened with water to clean the external case parts. Do not attempt to clean internally.

### **Product Markings**

CE - the CE mark is a registered trademark of the European Community. A CE mark accompanied by a year indicated the year the design was proven.

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

### **CERTIFICATION**

Hewlett-Packard Company certifies that this product met its published specifications at the time of shipment from the factory.

Hewlett-Packard further certifies that its calibration measurements are traceable to the United States National Institute of Standards and Technology, to the extent allowed by the Institute's calibration facility, and to the calibration facilities of other International Standards Organization members.

---

## **Warranty**

This Hewlett-Packard instrument product is warranted against defects in material and workmanship for a period of one year from date of shipment. During the warranty period, Hewlett-Packard Company will at its option, either repair or replace products which prove to be defective.

For warranty service or repair, this product must be returned to a service facility designated by HP. Buyer shall prepay shipping charges to HP and HP shall pay shipping charges, duties, and taxes for products returned to HP from another country.

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

### **LIMITATION OF WARRANTY**

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

NO OTHER WARRANTY IS EXPRESSED OR IMPLIED. HP SPECIFICALLY DISCLAIMS THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE.

### **EXCLUSIVE REMEDIES**

THE REMEDIES PROVIDED HEREIN ARE BUYER'S SOLE AND EXCLUSIVE REMEDIES. HP SHALL NOT BE LIABLE FOR ANY DIRECT, INDIRECT, SPECIAL, INCIDENTAL, OR CONSEQUENTIAL DAMAGES, WHETHER BASE ON CONTRACT, TORT, OR ANY OTHER LEGAL THEORY.



---

## **ASSISTANCE**

### **Maintenance Agreements**

Product maintenance agreements and other customer assistance agreements are available for Hewlett-Packard products. For any assistance, contact your nearest Hewlett-Packard Sales and Service Office.

## Regional Sales Offices

**IMPORTANT**

### Regional Sales and Service Offices

<p>Eastern USA Sales Office Hewlett-Packard Company 2101 Gather Rd. Rockville, MD 20850 Tel: (301) 258-2000</p>	<p>Eastern USA Sales Office Hewlett-Packard Company 2101 Gather Rd. Rockville, MD 20850 Tel: (301) 258-2000</p>	<p>Midwestern USA Sales and Service Hewlett-Packard Company 5201 Tollview Drive Rolling Meadows, IL 60008 Tel: (708) 342-2000</p>
<p>Southern USA Sales and Service Hewlett-Packard Company 1995 North Park Place Atlanta, GA 30339 Sales Tel: (404) 955-1500 Fax: (404) 980-7292 Service Tel: (404) 850-2544 Fax: (404) 980-7292</p>	<p>Southern USA Service Center Hewlett-Packard Company 930 E. Campbell Road Richardson, TX 75081 Tel: (214) 699-4331</p>	<p>Western USA Service Center Hewlett-Packard Company 301 E. Evelyn Avenue Mountain View, CA 94041 Tel: (415) 694-2000 Fax: (415) 694-0601</p>
<p>Western USA Sales and Service Hewlett-Packard Company 24 Inverness Place East Englewood, CO 80112 Sales Tel: (303) 649-5000 Fax: (303) 649-5787 Service Tel: (303) 649-5512 Fax: (303) 649-5787</p>	<p>Western USA Sales and Service Hewlett-Packard Company 1421 South Manhattan Avenue Fullerton, CA 92631 Sales Tel: (714) 999-6700 Fax: (714) 778-3033 Service Tel: (714) 758-5490 Fax: (714) 778-3033</p>	<p>United States of America Customer Information Center Hewlett-Packard Company Tel: (800) 752-0900 6:00 am to 5:00 pm Pacific Time Parts Direct: 1-800-227-8164</p>
<p>South Eastern Europe Sales and Service Hewlett-Packard Ges. m.b.h. Liebigasse 1 P.O. Box 72 A-1222 Vienna, Austria Telephone: 43 222 2500 0 Telex: 13 4425</p>	<p>European Multicountry Region Sales and Service Hewlett-Packard S.A. P.O. Box 95 150, Route dv Nant_dl_AVRIL CH-1217 Meyrin 2 Geneva, Switzerland Telephone: (41/22) 780-8111 Fax: (41/22) 780-8542</p>	<p>Northern Europe Sales and Service Hewlett-Packard Nederland B.V. Startbaan 16 1187 XR Amstelveen, The Netherlands P.O. Box 667 Telephone: 31/20 5476911 X 6631 Fax: 31-20-6471825NL</p>

<p>Asia Sales and Service Hewlett-Packard Asia Ltd. 22-30/F Peregrine Tower Lippo Center 89 Queensway, Central Hong Kong G.P.O. Box 863 Hong Kong Telephone: 852-848-7777 Fax: 852-868-4997</p>	<p>Japan Sales and Service Yokogawa-Hewlett-Packard Ltd. 3-29-21, Takaido-Higashi Suginami-Ku, Tokyo 168 Telephone: 81 3 3331-6111 Fax: 81 3 3331-6631</p>	<p>International Sales Branch Headquarters Sales and Service Hewlett-Packard S.A. 39 Rue Veyrot P.O. Box 365 1217 Meyrin 1 Geneva, Switzerland Telephone: 41-22-780-4111 Fax: 41-22-780-4770</p>
<p>Australia, New Zealand Sales and Service Hewlett-Packard Ltd. P.O. Box 221 31-41 Joseph Street Blackburn, Victoria 3130 Telephone: (61/3) 895-2895 Fax: (61/3) 898-9257</p>	<p>Canada Sales and Service Hewlett-Packard (Canada) Ltd. 5150 Spectrum Way Mississauga, Ontario L4W 5G1 Canada Telephone: (416) 206-4725 Fax: (416) 206-4739</p>	<p>Canada Service Center Hewlett-Packard Company 17500 Transcanada Highway S. Serv Road Kirkland, Quebec H9J 2X8 Canada Telephone: (416) 206-3295</p>
<p>Canada Service Center Hewlett-Packard Ltd. 11120 178 Street Edmonton, Alberta T5S 1P2 Canada Telephone: (403) 486-6666 Fax: (403) 489-8764</p>	<p>Latin America Hewlett-Packard Company LAHQ Mexico City Col. Lomas de Virreyes 11000 Mexico D.F. Mexico Telephone: (52/5) 326-4000 Fax: (52/5) 202 7718</p>	<p>United Kingdom Sales and Service Hewlett-Packard Ltd. Cain Road Amen Corner Bracknell, Berkshire RG12 1HN United Kingdom Telephone: 44 344 360000</p>



<b>In This Manual</b> .....	<b>15</b>
<b>1. Using HP-IB</b>	
Overview of the Test Set .....	18
Getting Started .....	25
Remote Operation .....	35
Addressing .....	37
IEEE 488.1 Remote Interface Message Capabilities .....	38
Remote/Local Modes .....	44
<b>2. Methods For Reading Measurement Results</b>	
Background .....	48
HP BASIC 'ON TIMEOUT' Example Program .....	50
HP BASIC 'MAV' Example Program .....	54
<b>3. HP-IB Command Guidelines</b>	
Sequential and Overlapped Commands .....	60
Guidelines for Operation .....	61
<b>5. HP-IB Command Dictionary</b>	
Notice .....	96
Using This Dictionary .....	97
:AFANalyzer subsystem .....	98
:AFG1   AFGENERATOR1 subsystem .....	106
:AFG2   AFGENERATOR2 subsystem .....	108
:CANalyzer subsystem .....	110
:CBUFFER subsystem .....	120
:CCOMmon subsystem .....	123
:CDANalyzer subsystem .....	124
:CGENERator subsystem .....	132
:CONFigure subsystem .....	135
:DISPlay subsystem .....	159
Integer Number Setting Syntax .....	161
:MEASure subsystem .....	162
Multiple Real Number Setting Syntax .....	186
Number Measurement Syntax .....	187
:OSCilloscope subsystem .....	190
:PROGram subsystem .....	201
Real Number Setting Syntax .....	203
[:REGister] subsystem .....	205
:RFANalyzer subsystem .....	208
:RFGeneraTOR subsystem .....	212
:SANalyzer subsystem .....	216
:STATus .....	225
:SYSTem .....	234
:TESTs Subsystem .....	235
:TRIGger subsystem .....	241

<b>6. IEEE Common Commands</b>	
IEEE 488.2 Common Commands .....	244
Common Command Descriptions .....	245
<b>7. Triggering Measurements</b>	
Triggering Measurements .....	256
<b>8. Advanced Operations</b>	
Increasing Measurement Throughput .....	264
Status Reporting .....	270
HP-IB Service Requests .....	317
Instrument Initialization .....	326
Passing Control .....	337
<b>9. Memory Cards/Mass Storage</b>	
Default File System .....	346
Mass Storage Device Overview .....	347
Default Mass Storage Locations .....	352
Mass Storage Access .....	354
DOS and LIF File System Considerations .....	355
Using the ROM Disk .....	361
Using Memory Cards .....	362
Backing Up Procedure and Library Files .....	367
Copying Files Using IBASIC Commands .....	368
Using RAM Disk .....	370
<b>10. IBASIC Controller</b>	
Introduction .....	374
The IBASIC Controller Screen .....	375
Important Notes for Program Development .....	377
Program Development .....	378
Interfacing to the IBASIC Controller using Serial Ports .....	380
Choosing Your Development Method .....	394
Method #1. Program Development on an External BASIC Language Computer .....	396
Method #2. Developing Programs on the Test Set Using the IBASIC EDIT Mode .....	403
Method #3. Developing Programs Using Word Processor on a PC (Least Preferred) .....	407
Uploading Programs from the Test Set to a PC .....	415
Serial I/O from IBASIC Programs .....	416
PROGram Subsystem .....	419
The TESTS Subsystem .....	441
<b>11. Error Messages</b>	
General Information About Error Messages .....	448
HP-IB Errors .....	454
Text Only HP-IB Errors .....	455

Numbered HP-IB Error Descriptions . . . . .	457
---	-----





---

## In This Manual

This manual explains how to use the HP 8935 to manually test a CDMA Base Station.

### What is Discussed This Manual

This document presents a step-by-step approach to CDMA base station testing using the Test Set, including what you need to know before you can start testing.

### What is Not Discussed in this Manual

General operation of the Test Set, such as changing display screens and their associated controls, is discussed in the Reference Guide (HP part number E6380-90019).

Detailed operation of the Test Set's spectrum analyzer and oscilloscope. Although there are basic explanations in this manual, more detail is provided in the Reference Guide.

How to control your base station, switch system, or any other software or hardware associated with your cell site equipment. Each manufacturer and cellular service provider has their own cell site control and base station configuration procedures that go beyond the scope of this documentation.

How to perform IBASIC programming operations, such as writing, editing, copying, or cataloguing programs. Programming the Test Set is explained in the Programming Manual (HP part number E6380-90018), and the IBASIC language is explained in the HP Instrument BASIC User's Handbook (HP part number E2083-90005).

### Conventions Used in This Manual

The following conventions are used throughout this manual to help clarify instructions and reduce unnecessary text:

**Test Set** refers to the HP 8935.

Test Set keys are indicated like this: **Preset**

Test Set screen information, such as a measurement result or an error message, is shown like this: TX Channel Power -1.3 dBm

**Which Document is Required?** The following documents are part of the HP 8935 document set. Use the table to help you decide which document you need.

**Table 1 Document Navigation**

<b>Document</b>	<b>Part Number</b>	<b>Usage</b>
CDMA Application Guide	E6380-90016	Use this manual for basic CDMA measurements and for getting started with the Test Set.
AMPS Application Guide	E6380-90017	Use this manual for making AMPS base station measurements.
Reference Guide	E6380-90019	Use this manual for screen and field descriptions and general operation information about the Test Set.
Programmer's Guide	E6380-90018	Use this manual to learn HP-IB syntax and for learning how to program the Test Set.
Assembly Level Repair Guide	E6380-90015	Use this manual to perform calibration on the Test Set and for general service information.

---

# **1** **Using HP-IB**

## Overview of the Test Set

The Test Set combines numerous separate test instruments and an Instrument BASIC (IBASIC) Controller into one package. All of the Test Set's functions can be automatically controlled through application programs running on the built-in IBASIC Controller or on an external controller connected through HP-IB.

Developing programs for the Test Set is simplified if the programmer has a basic understanding of how the Test Set operates. An overview of the Test Set's operation is best presented in terms of how information flows through the unit. The simplified block diagrams shown in [figure 1-1 on page 23](#) and [figure 1-2 on page 24](#) depict how instrument control information and measurement result information are routed among the Test Set's instruments, instrument control hardware, built-in IBASIC controller, and other components.

The Test Set has two operating modes: Manual Control mode and Automatic Control mode. In Manual Control mode the Test Set's operation is controlled through the front panel keypad/rotary knob. There are two Automatic Control modes: Internal and External. In Internal Automatic Control mode the Test Set's operation is controlled by an application program running on the built-in IBASIC Controller. In External Automatic Control mode the Test Set's operation is controlled by an external controller connected to the Test Set through the HP-IB interface.

## Manual Control Mode

The Test Set's primary instruments are shown on the left side of [figure 1-1](#). There are two classes of instruments in the Test Set: signal analyzers (RF Analyzer, AF Analyzer, Oscilloscope, Spectrum Analyzer) and signal sources (RF Generator, AF Generator #1, AF Generator #2).

Since so many instruments are integrated into the Test Set, it is not feasible to have an actual "front panel" for each instrument. Therefore, each instrument's front panel is maintained in firmware and is displayed on the display whenever the instrument is selected. Only one instrument front panel can be displayed at any given time (up to four measurement results can be displayed simultaneously if desired). Just as with stand alone instruments, instrument front panels in the Test Set can contain instrument setting information, measurement result(s), or data input from the DUT (Device Under Test).

Using the Test Set in Manual Control mode is analogous to using a set of bench or rack-mounted test equipment. To obtain a measurement result with a bench or racked system, the desired measurement must be "active." For example, if an RF power meter is in the bench or racked system and the user wishes to measure the power of an RF carrier they must turn the power meter on, and look at the front panel to see the measurement result. Other instruments in the system may be turned off but this would not prevent the operator from measuring the RF power.

Conceptually, the same is true for the Test Set. In order to make a measurement or input data from a DUT, the desired measurement field or data field must be "active." This is done by using the front panel keypad/rotary knob to select the instrument whose front panel contains the desired measurement or data field and making sure that the desired measurement or data field is turned ON.

[Figure 1-1](#) shows that instrument selection is handled by the screen control hardware which routes the selected instrument's front panel to the display. Once an instrument's front panel is displayed on the display, the user can manipulate the instrument settings, such as turning a specific measurement or data field on or off, using the keypad/rotary knob. [Figure 1-1](#) also shows that instrument setup is handled by the Instrument Control hardware which routes setup information from the front panel to the individual instruments.

An HP-IB/RS-232/Parallel Printer interface is located in the Test Set. In Manual Control mode this provides the capability of connecting an external HP-IB, serial, or parallel printer to the Test Set so that display screens can be printed.

## Internal Automatic Control Mode

In Internal Automatic Control mode the Test Set's operation is controlled by an application program running on the built-in Instrument BASIC (IBASIC) Controller. The built-in controller runs programs written in IBASIC, a subset of the Hewlett-Packard BASIC programming language used on the HP 9000 Series 200/300 System Controllers. IBASIC is the only programming language supported on the built-in IBASIC Controller.

### Similarities Between the Test Set's IBASIC Controller and Other Single-Tasking Controllers

The architecture of the IBASIC Controller is similar to that of other single-tasking instrumentation controllers. Only one program can be run on the IBASIC Controller at any given time. The program is loaded into RAM memory from some type of mass storage device. Four types of mass storage devices are available to the Test Set: SRAM memory cards, ROM memory cards connected to the HP-IB interface, internal RAM disc, and internal ROM disc. Three types of interfaces are available for connecting to external instruments and equipment: HP-IB, RS-232, and 16-bit parallel.

**Figure 1-2** shows how information is routed inside the Test Set when it is in Internal Automatic Control mode. In Manual Control mode certain Test Set resources are dedicated to manual operation. These resources are switched to the IBASIC Controller when an IBASIC program is running. These include the serial interface ports at select codes 9, 10, and 11, the HP-IB interface at select code 7, the parallel printer interface ports at select codes 15 and 16, and the display. In Manual Control mode, front panel information (instrument settings, measurement results, data input from the DUT) is routed to the display through the screen control hardware. In Internal Automatic Control mode the measurement results and data input from the DUT are routed to the IBASIC Controller through a dedicated HP-IB interface. Also, in Internal Automatic Control mode, the display is dedicated to the IBASIC Controller for program and graphics display. This means instrument front panels cannot be displayed on the display when an IBASIC program is running.

### Differences Between the Test Set's IBASIC Controller and Other Single-Tasking Controllers

The IBASIC Controller is unlike other single tasking instrumentation controllers in several ways. First, it does not have a keyboard. This imposes some limitations on creating and editing IBASIC programs directly on the Test Set. In Internal Automatic Control mode a "virtual" keyboard is available in firmware which allows the operator to enter alphanumeric data into a dedicated input field using the rotary knob. This is not the recommended programming mode for the IBASIC Controller. This feature is provided to allow user access to IBASIC

programs for short edits or troubleshooting. Several programming modes for developing IBASIC programs to run on the internal IBASIC Controller are discussed in this manual.

Secondly, the IBASIC Controller has a dedicated HP-IB interface, select code 8 in [figure 1-2](#), for communicating with the internal instruments of the Test Set. This HP-IB interface is only available to the IBASIC Controller. There is no external connector for this HP-IB interface. No external instruments may be added to this HP-IB interface. The HP-IB interface, select code 7 in [figure 1-2](#), is used to interface the Test Set to external instruments or to an external controller. The dedicated HP-IB interface at select code 8 conforms to the IEEE 488.2 Standard in all respects but one. The difference being that each instrument on the bus does not have a unique address. The Instrument Control Hardware determines which instrument is being addressed through the command syntax. Refer to [Chapter 5, "HP-IB Command Dictionary"](#) for a listing of the HP-IB command syntax for the Test Set.

## External Automatic Control Mode

In External Automatic Control mode the Test Set's operation is controlled by an external controller connected to the Test Set through the HP-IB interface. When in External Automatic Control mode the Test Set's internal configuration is the same as in Manual Control Mode with two exceptions:

1. Configuration and setup commands are received through the external HP-IB interface, select code 7, rather than from the front-panel keypad/rotary knob.
2. The MEASure command is used to obtain measurement results and DUT data through the external HP-IB interface.

[Figure 1-1 on page 23](#) shows how information is routed inside the Test Set in Manual Control mode. [Figure 1-1](#) also shows that certain Test Set resources are dedicated to the IBASIC Controller (Memory Card, ROM disk, Serial Ports 10 and 11) and are not directly accessible to the user in Manual Control Mode. In addition, [figure 1-1](#) shows that Serial Port 9 and Parallel Printer Port 15 are accessible as write-only interfaces for printing in Manual Control mode. These same conditions are true when in External Automatic Control mode. If the user wished to access these resources from an external controller, an IBASIC program would have to be run on the Test Set from the external controller.

## Writing programs for the Test Set

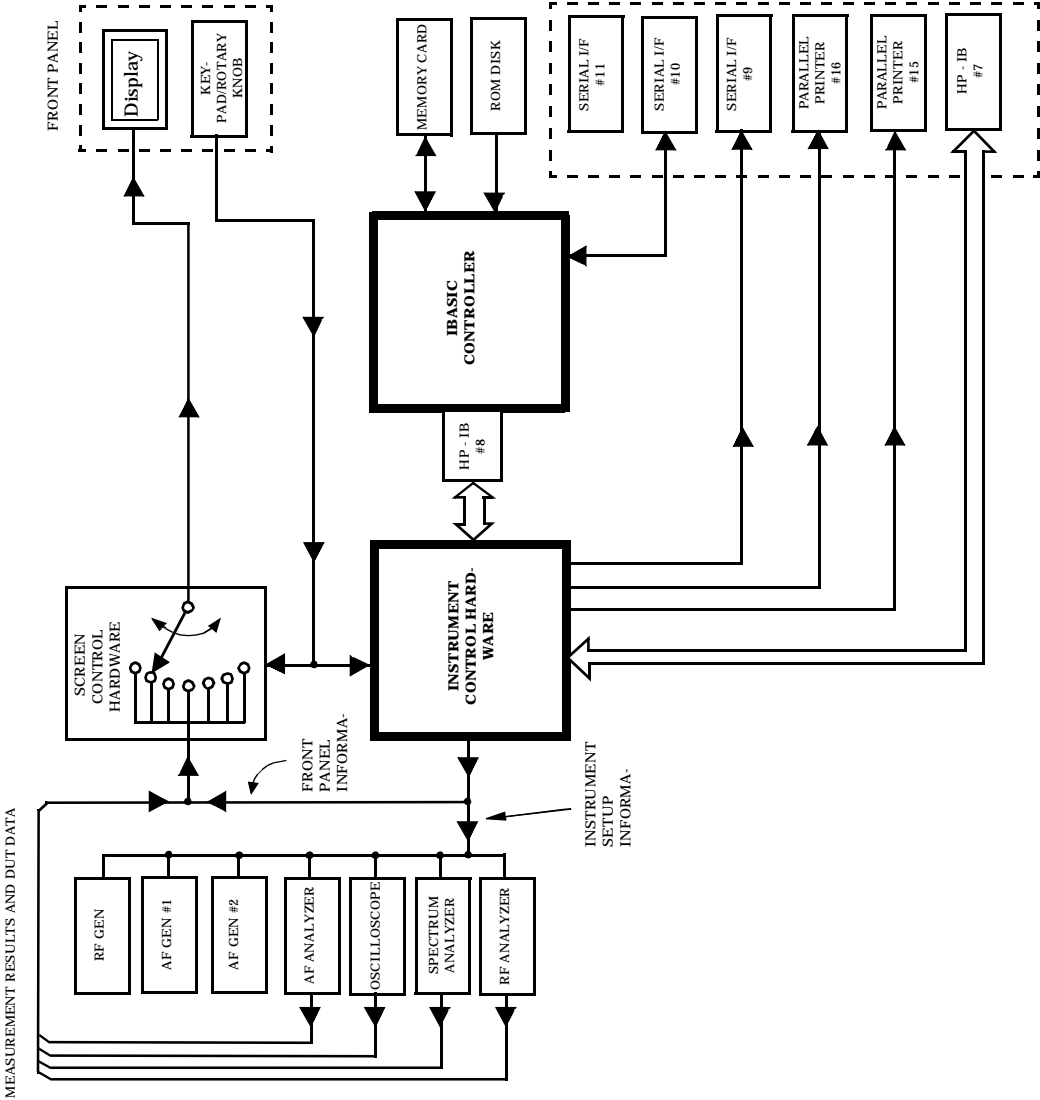
One of the design goals for automatic control of the Test Set was that it operate the same way programmatically as it does manually. This is a key point to remember when developing programs for the Test Set. The benefit of this approach is that to automate a particular task, one need only figure out how to do the task manually and then duplicate the same process in software. This has several implications when designing and writing programs for the Test Set:

1. In Manual Control mode a measurement must be “active” in order to obtain a measurement result or input data from the DUT. From a programming perspective this means that before attempting to read a measurement result or to input data from the DUT, the desired screen for the measurement result or data field must be selected using the DISPLAY command and the field must be in the ON state.
2. In Manual Control mode instrument configuration information is not routed through the screen control hardware. From a programming perspective this means that configuration information can be sent to any desired instrument without having to first select the instrument’s front panel with the DISPLAY command.

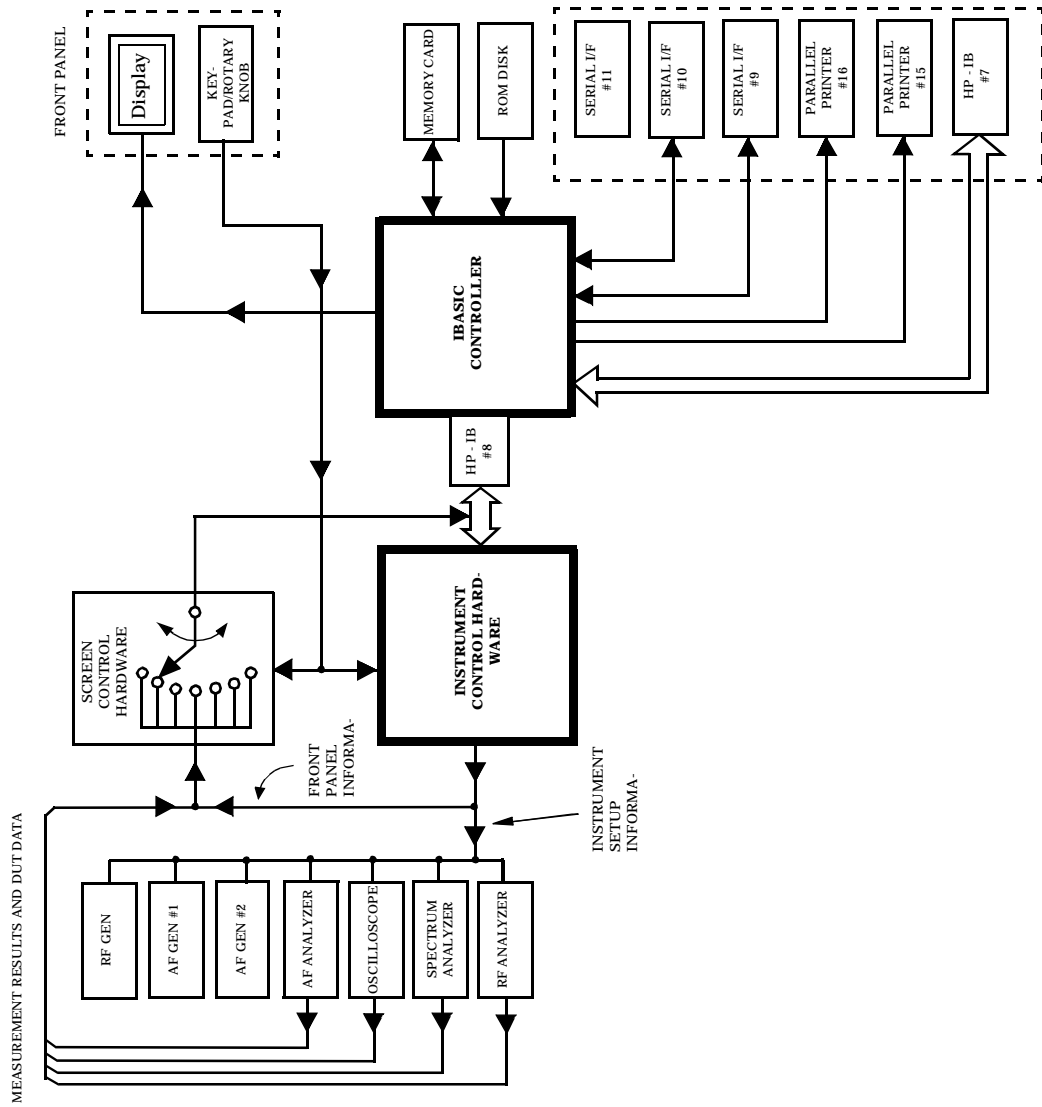
Keeping these points in mind during program development will minimize program development time and reduce problems encountered when running the program.



Figure 1-1 Manual Control Mode



**Figure 1-2 Internal Automatic Control Mode**



---

## Getting Started

### What is HP-IB?

The Hewlett-Packard Interface Bus (HP-IB) is Hewlett-Packard's implementation of the IEEE 488.1-1987 Standard Digital Interface for Programmable Instrumentation. Incorporation of the HP-IB into the Test Set provides several valuable capabilities:

- Programs running in the Test Set's IBASIC Controller can control all the Test Set's functions using its internal HP-IB. This capability provides a single-instrument automated test system. (The HP 11807 Radio Test Software utilizes this capability.)
- Programs running in the Test Set's IBASIC Controller can control other instruments connected to the external HP-IB.
- An external controller, connected to the external HP-IB, can remotely control the Test Set.
- An HP-IB printer, connected to the external HP-IB, can be used to print test results and full screen images.

### HP-IB Information Provided in This Manual

#### What Is Explained

- How to configure the Test Set for HP-IB operation
- How to make an instrument setting over HP-IB
- How to read-back instrument settings over HP-IB
- How to make measurements over HP-IB
- How to connect external PCs, terminals or controllers to the Test Set
- HP-IB command syntax for the Test Set
- IBASIC program development
- IBASIC program transfer over HP-IB
- Various advanced functions such as, increasing measurement throughput, status reporting, error reporting, pass control, and so forth

## What Is Not Explained

- HP-IB (IEEE 488.1, 488.2) theory of operation<sup>1</sup>
- HP-IB electrical specifications\*
- HP-IB connector pin functions\*
- IBASIC programming (other than general guidelines related to HP-IB)<sup>2</sup>

## General HP-IB Programming Guidelines

The following guidelines should be considered when developing programs which control the Test Set through HP-IB:

- Guideline #1.
  1. Bring the Test Set to its preset state using the front-panel PRESET key. This initial step allows you to start developing the measurement sequence with most fields in a known state.
  2. Make the measurement manually using the front-panel controls of the Test Set. Record, in sequential order, the screens selected and the settings made within each screen. The record of the screens selected and settings made in each screen becomes the measurement procedure.
  3. Record the measurement result(s).
  4. Develop the program using the measurement procedure generated in [step 2](#). Be sure to start the programmatic measurement sequence by bringing the Test Set to its preset state using the \*RST Common Command. As the measurement procedure requires changing screens, use the DISPlay command to select the desired screen followed by the correct commands to set the desired field(s).

---

### NOTE

When IBASIC programs are running the display is dedicated to the IBASIC Controller for program and graphics display. This means instrument front panels are not displayed when an IBASIC program is running. However, the DISPlay <screen> command causes all setting and measurement fields in the <screen> to be accessible programmatically. Attempting to read from a screen that has not been made accessible by the DISPlay command will cause  
HP-IB Error:-420 Query UNTERMINATED, or  
HP-IB Error: -113 Undefined header

---

1. Refer to the *Tutorial Description of the Hewlett-Packard Interface Bus* (HP P/N 5952-0156) for detailed information on HP-IB theory and operation.
2. Refer to the *HP Instrument BASIC Users Handbook* (HP P/N E2083-90005) for more information on the IBASIC Version 2.0 language.

Make sure the desired measurement is in the ON state. This is the preset state for most measurements. However, if a previous program has set the state to OFF, the measurement will not be available. Attempting to read from a measurement field that is not in the ON state will cause HP-IB Error: -420 Query UNTERMINATED.

5. If the trigger mode has been changed, trigger a reading.

---

NOTE

Triggering is set to FULL SETTling and REPetitive RETRiggering after receipt of the \*RST Common Command. These settings cause the Test Set to trigger itself and a separate trigger command is not necessary.

---

6. Send the MEASure query command to initiate a reading. This will place the measured value into the Test Set's Output Queue.

---

NOTE

When making AF Analyzer SINAD, Distortion, Signal to Noise Ratio, AF Frequency, DC Level, or Current measurements, the measurement type must first be selected using the SElect command. For example, MEAS:AFR:SEL'SINAD' followed by MEAS:AFR:SINAD?

---

7. Use the ENTER statement to transfer the measured value to a variable within the context of the program.

The following example program illustrates how to make settings and then take a reading from the Test Set. This setup takes a reading from the spectrum analyzer marker after tuning it to the RF generator's output frequency.

#### Example

```
10 Addr=714
20 OUTPUT Addr;"*RST" !Preset to known state
30 OUTPUT Addr;"TRIG:MODE:RETR SING" !Sets single trigger
40 OUTPUT Addr;"DISP RFG" !Selects the RF Gen screen
50 OUTPUT Addr;"AFG1:FM:STAT OFF" !Turns FM OFF
60 OUTPUT Addr;"RFG:AMPL -66 DBM" !Sets RF Gen ampl to -66 dBm
70 OUTPUT Addr;"RFG:FREQ 500 MHZ" !Sets RF Gen freq to 500 MHz
80 OUTPUT Addr;"RFG:AMPL:STAT ON" !Turns RF Gen output ON
90 OUTPUT Addr;"DISP SAN"!Selects Spectrum Analyzer's screen
100 OUTPUT Addr;"SAN:CRF 500 MHZ" !Center Frequency 500 MHz
110 ! -----MEASUREMENT SEQUENCE-----
120 OUTPUT Addr;"TRIG" !Triggers reading
130 OUTPUT Addr;"MEAS:SAN:MARK:LEV?" !Query of Spectrum
140 !Analyzer's marker level
150 ENTER Addr;Lvl !Places measured value in variable Lvl
160 DISP Lvl!Displays value of Lvl
170 END
```

The RF Generator's output port and the Spectrum Analyzer's input port are preset to the RF IN/OUT port. This allows the Spectrum Analyzer to measure the RF Generator with no external connections. The Spectrum Analyzer marker is always tuned to the center frequency of the Spectrum Analyzer after preset. With the RF Generator's output port and Spectrum Analyzer input port both directed to the RF IN/OUT port, the two will internally couple with 46 dB of gain, giving a measured value of approximately -20 dBm. While not a normal mode of operation this setup is convenient for demonstration since no external cables are required. This also illustrates the value of starting from the preset state since fewer programming commands are required.

- **Guideline #2.**

If the program stops or “hangs up” when trying to ENTER a measured value, it is most likely that the desired measurement field is not available. There are several reasons that can happen:

1. The screen where the measurement field is located has not been DISPLAYed before querying the measurement field.
2. The measurement is not turned ON.
3. The squelch control is set too high. If a measurement is turned ON but is not available due to the Squelch setting, the measurement field contains four dashes (- - - -). This is a valid state. The Test Set is waiting for a signal of sufficient strength to unsquelch the receiver before making a measurement. If a measurement field which is squelched is queried the Test Set will wait indefinitely for the receiver to unsquelch and return a measured value.
4. The RF Analyzer's Input Port is set to ANT (antenna) while trying to read TX power. TX power is not measurable with the Input Port set to ANT. The TX power measurement field will display four dashes (- - - -) indicating the measurement is unavailable.
5. The input signal to the Test Set is very unstable causing the Test Set to continuously autorange. This condition will be apparent if an attempt is made to make the measurement manually.
6. Trigger mode has been set to single trigger (TRIG:MODE:RETRig SINGLE) and a new measurement cycle has not been triggered before attempting to read the measured value.
7. The program is attempting to make an FM deviation or AM depth measurement while in the RX TEST screen. FM or AM measurements are not available in the RX TEST screen. FM or AM measurements are made from the AF Analyzer screen by setting the AF An1 In field to FM or AM Demod.

- **Guideline #3.**

The syntax diagrams in [chapter 5, "HP-IB Command Dictionary,"](#) show where single quotes are needed and where spaces are needed.

**Example**

```
OUTPUT 714; "DISP<space>AFAN"
```

```
OUTPUT 714; "AFAN:DEMP<space>'Off' "
```

**Improper use of single quotes and spaces will cause,**  
HP-IB Error:-103 Invalid Separator

- **Guideline #4.**

When making settings to fields that can be turned OFF with the STATE ON/OFF command (refer to the [Chapter 5, "HP-IB Command Dictionary"](#)), make sure the STATE is ON if the program uses that field. Note that if the STATE is OFF, just setting a numeric value in the field will not change the STATE to ON. This is different than front-panel operation whereby the process of selecting the field and entering a value automatically sets the STATE to ON. Programmatically, fields must be explicitly set to the ON state if they are in the OFF state.

For example, the following command line would set a new AMPS ENCoder SAT tone deviation and then turn on the SAT tone (note the use of the ; to back up one level in the command hierarchy so that more than one command can be executed in a single line):

**Example**

```
OUTPUT 714; "ENC:AMPS:SAT:FM 2.1 KHZ;FM:STAT ON"
```

To just turn on the SAT tone without changing the current setting the following commands would be used:

**Example**

```
OUTPUT 714; "ENC:AMPS:SAT:FM:STAT ON"
```

## Control Annunciators

The letters and symbols at the top right corner of the display indicate these conditions:

- R indicates the Test Set is in remote mode. The Test Set can be put into the remote mode by an external controller or by an IBASIC program running on the built-in IBASIC controller.
- L indicates that the Test Set has been addressed to Listen.
- T indicates that the Test Set has been addressed to Talk.
- S indicates that the Test Set has sent the Require Service message by setting the Service Request (SRQ) bus line true. (See "[Status Reporting](#)" on page 270.)
- C indicates that the Test Set is currently the Active Controller on the bus.
- \* indicates that an IBASIC program is running.
- ? indicates that an IBASIC program is waiting for a user response.
- - indicates that an IBASIC program is paused.



## Preparing the Test Set For HP-IB Use

1. If other HP-IB devices are in the system, attach an HP-IB cable from the Test Set's HP-IB connector to any one of the other devices in the test system.
2. Access the I/O CONFIGURE screen and perform the following steps:
  - a. Set the Test Set's HP-IB address using the `HP-IB Adrs` field.
  - b. Set the Test Set's HP-IB Controller capability using the `Mode` field.
    - `Talk&Listen` configures the Test Set to *not* be the System Controller. The Test Set has Active Controller capability (take control/pass control) in this mode. Use this setting if the Test Set will be controlled through HP-IB from an external controller.
    - `Control` configures the Test Set to be the System Controller. Use this setting if the Test Set will be the only controller on the HP-IB. Selecting the Control mode automatically makes the Test Set the Active Controller.

---

### NOTE

---

Only one System Controller can be configured in an HP-IB system. Refer to "[Passing Control](#)" on page 337 for further information.

3. If an HP-IB printer is or will be connected to the Test Set's HP-IB connector then,
  - a. access the PRINT CONFIGURE screen.
  - b. select one of the supported HP-IB printer models using the `Model` field.
  - c. set the `Printer Port` field to HP-IB.
  - d. set the printer address using the `Printer Address` field.

## Using the HP-IB with the Test Set's built-in IBASIC Controller

The Test Set has two HP-IB interfaces, an internal-only HP-IB at select code 8 and an external HP-IB at select code 7. The HP-IB at select code 8 is only available to the built-in IBASIC Controller and is used exclusively for communication between the IBASIC Controller and the Test Set. The HP-IB at select code 7 serves three purposes:

1. It allows the Test Set to be controlled by an external controller
2. It allows the Test Set to print to an external HP-IB printer
3. It allows the built-in IBASIC Controller to control external HP-IB devices

IBASIC programs running on the Test Set's IBASIC Controller must use the internal-only HP-IB at select code 8 to control the Test Set. IBASIC programs would use the external HP-IB at select code 7 to control HP-IB devices connected to the HP-IB connector.

---

### NOTE

---

Refer to "[Overview of the Test Set](#)" on page 18 for a detailed explanation of the Test Set's architecture.

When using a BASIC language Workstation with an HP-IB interface at select code 7 to control the Test Set, HP-IB commands would look like this:

#### Example

```
! This command is sent to the Test Set at address 14.  
OUTPUT 714;"*RST"  
  
! This command is sent to another instrument whose address is 19.  
OUTPUT 719;"*RST"
```

When executing the same commands on the Test Set's IBASIC Controller, the commands would look like this:

#### Example

```
OUTPUT 814;"*RST"  
  
! Command sent to internal-only HP-IB at select code 8,  
! Test Set's address does not change  
OUTPUT 719;"*RST"  
  
! Command sent to external HP-IB at select code 7,  
! other instrument's address does not change.
```

## Basic Programming Examples

The following simple examples illustrate the basic approach to controlling the Test Set through the HP-IB. The punctuation and command syntax used for these examples is given in [Chapter 5, "HP-IB Command Dictionary"](#).

The bus address 714 used in the following BASIC language examples assumes an HP-IB interface at select code 7, and a Test Set HP-IB address of 14. All examples assume an external controller is being used.

### To Change a Field's Setting over HP-IB

1. Use the DISPLAY command to access the screen containing the field whose setting is to be changed.
2. Make the desired setting using the proper command syntax (refer to [Chapter 5, "HP-IB Command Dictionary"](#) for proper syntax).

The following example makes several instrument setting changes:

#### Example

```
OUTPUT 714;"DISP RFG" !Display the RF Generator screen.
OUTPUT 714;"RFG:FREQ 850 MHZ" !Set the RF Gen Freq to 850 MHz.
OUTPUT 714;"RFG:OUTP 'DUPL'"!Set the Output Port to Duplex.
OUTPUT 714;"DISP AFAN"!Display the AF Analyzer screen.
OUTPUT 714;"AFAN:INP 'FM DEMOD'"!Set the AF Anl In to FM Demod.
```

### To Read a Field's Setting over HP-IB37

1. Use the DISPLAY command to access the screen containing the field whose setting is to be read.
2. Use the Query form of the syntax for that field to place the setting value into the Test Set's output buffer.
3. Enter the value into the correct variable type within the program context (refer to [Chapter 5, "HP-IB Command Dictionary"](#), for proper variable type).

The following example reads several fields.

#### Example

```
OUTPUT 714;"DISP AFAN"!Display the AF Analyzer screen.
OUTPUT 714;"AFAN:INP?"!Query the AF Anl In field
ENTER 714;Af_input$ !Enter returned value into a string ariable.
OUTPUT 714;"DISP RFG"!Display the RF Generator screen
OUTPUT 714;"RFG:FREQ?"!Query the RF Gen Frequency field.
ENTER 714;Freq !Enter the returned value into a numeric variable
```

---

**NOTE**

When querying measurements or settings through HP-IB, the Test Set always returns numeric values in HP-IB Units or Attribute Units, regardless of the current Display Units setting. Refer to "[HP-IB Units \(UNITS\)](#)" on page 68 and "[Attribute Units \(AUNits\)](#)" on page 70 for further information.

---

### **To Make a Simple Measurement**

The basic method for making a measurement is very similar to the method used to read a field setting.

1. Use the DISPlay command to access the screen containing the desired measurement.
2. Use the MEASure form of the syntax for that measurement to place the measured value into the Test Set's output buffer.
3. Enter the value into the correct variable type within the program context (refer to [Chapter 5, "HP-IB Command Dictionary"](#) for proper variable type).

The following example measures the power of an RF signal.

#### **Example**

```
!Display the RF Analyzer screen.  
OUTPUT 714;"DISP RFAN"  
!Measure the RF power and place result in output buffer.  
OUTPUT 714;"MEAS:RFR:POW?"  
!Enter the measured value into a numeric variable.  
ENTER 714;Tx_power
```

The above example is very simple and is designed to demonstrate the fundamental procedure for obtaining a measurement result. Many other factors must be considered when designing a measurement procedure, such as instrument settings, signal routing, settling time, filtering, triggering and measurement speed.

---

## Remote Operation

The Test Set can be operated remotely through the Hewlett-Packard Interface Bus (HP-IB). Except as otherwise noted, the Test Set complies with the IEEE

488.1-1987 and IEEE 488.2-1987 Standards. Bus compatibility, programming and data formats are described in the following sections.

All front-panel functions, except those listed in [table 1-1](#), are programmable through HP-IB.

**Table 1-1 Non-Programmable Front Panel Functions**

Function	Comment
ON/OFF Power Switch	
Volume Control Knob	
Squelch Control Knob	The position of the Squelch Control knob cannot be programmed. However squelch can be programmed to either the Open or Fixed position. Refer to the Test Set's User's Guide for more information.
Cursor Control Knob	
SHIFT Key	
CANCEL Key	
YES Key	
NO Key	
ENTER Key	
Backspace (left-arrow) Key	
PREV Key	
HOLD (SHIFT, PREV Keys)	
PRINT (SHIFT, TESTS Keys)	
ADRS (SHIFT, LOCAL Keys)	
ASSIGN (SHIFT, k4 Keys)	
RELEASE (SHIFT, k5 Keys)	

## Remote Capabilities

### Conformance to the IEEE 488.1-1987 Standard

For all IEEE 488.1 functions implemented, the Test Set adheres to the rules and procedures as outlined in that Standard.

### Conformance to the IEEE 488.2-1987 Standard

For all IEEE 488.2 functions implemented, the Test Set adheres to the rules and procedures as outlined in that Standard with the exception of the \*OPC Common Command. Refer to the \*OPC Common Command description.

### IEEE 488.1 Interface Functions

The interface functions that the Test Set implements are listed in [table 1-2](#).

**Table 1-2 Test Set IEEE 488.1 Interface Function Capabilities**

Function	Capability
Talker	T6: No Talk Only Mode
Extended Talker	T0: No Extended Talker Capability
Listener	L4: No Listen Only Mode
Extended Listener	LE0: No Extended Listener Capability
Source Handshake	SH1: Complete Capability
Acceptor Handshake	AH1: Complete Capability
Remote/Local	RL1: Complete Capability
Service Request	SR1: Complete Capability
Parallel Poll	PP0: No Parallel Poll Capability
Device Clear	DC1: Complete Capability
Device Trigger	DT1: Complete Capability
Controller	C1: System Controller C3: Send REN C4: Respond to SRQ C11: No Pass Control to Self, No Parallel Poll
Drivers	E2: Tri-State Drivers

---

## Addressing

### Factory Set Address

The Test Set's HP-IB address is set to decimal 14 at the factory. The address can be changed by following the instructions in "[Setting the Test Set's Bus Address](#)" on page 37.

### Extended Addressing

Extended addressing (secondary command) capability is not implemented in the Test Set.

### Multiple Addressing

Multiple addressing capability is not implemented in the Test Set.

### Setting the Test Set's Bus Address

The Test Set's HP-IB bus address is set using the `HP-IB Adrs` field which is located on the I/O CONFIGURE screen. To set the HP-IB bus address; select the I/O CONFIGURE screen and position the cursor next to the `HP-IB Adrs` field. The address can be set from decimal 0 to 30 using the numeric DATA keys, or by pushing and then rotating the Cursor Control knob. There are no DIP switches for setting the HP-IB bus address in the Test Set. The new setting is retained when the Test Set is turned off.

### Displaying the Bus Address

The Test Set's HP-IB bus address can be displayed by pressing and releasing the SHIFT key, then the LOCAL key. The address is displayed in the upper left-hand corner of the display screen.

## **IEEE 488.1 Remote Interface Message Capabilities**

The remote interface message capabilities of the Test Set and the associated IEEE 488.1 messages and control lines are listed in [table 1](#).



Table 1 Test Set IEEE 488.1 Interface Message Capability

Message Type	Implemented	Response	IEEE 488.1 Message
Data	Yes	All front-panel functions, except those listed in <a href="#">table 1-2 on page 36</a> , are programmable. The Test Set can send status byte, message and setting information. All measurement results (except dashed “- - - -” displays) and error messages are available through the bus.	DAB END MTA MLA OTA
Remote	Yes	Remote programming mode is entered when the Remote Enable (REN) bus control line is true and the Test Set is addressed to listen. The R annunciator will appear in the upper-right corner of the display screen when the Test Set is in remote mode. All front-panel keys are disabled (except for the LOCAL key, POWER switch, Volume control and Squelch control knobs). When the Test Set enters remote mode the output signals and internal settings remain unchanged, except that triggering is reset to the state it was last set to in remote mode (Refer to <a href="#">"Triggering Measurements" on page 255</a> ).	REN MLA

Message Type	Implemented	Response	IEEE 488.1 Message
Local	Yes	The Test Set returns to local mode (full front-panel control) when either the Go To Local (GTL) bus command is received, the front-panel LOCAL key is pressed or the REN line goes false. When the Test Set returns to local mode the output signals and internal settings remain unchanged, except that triggering is reset to TRIG:MODE:SETT FULL;RETR REP. The LOCAL key will not function if the Test Set is in the local lockout mode.	GTL MLA
Local Lockout	Yes	Local Lockout disables all front-panel keys including the LOCAL key. Only the System Controller or the POWER switch can return the Test Set to local mode (front-panel control).	LLO
Clear Lockout/ Set Local	Yes	The Test Set returns to local mode (front-panel control) and local lockout is cleared when the REN bus control line goes false. When the Test Set returns to local mode the output signals and internal settings remain unchanged, except that triggering is set to TRIG:MODE:SETT FULL;RETR REP.	REN

<b>Message Type</b>	<b>Implemented</b>	<b>Response</b>	<b>IEEE 488.1 Message</b>
Service Request	Yes	The Test Set sets the Service Request (SRQ) bus line true if any of the enabled conditions in the Status Byte Register, as defined by the Service Request Enable Register, are true.	SRQ
Status Byte	Yes	The Test Set responds to a Serial Poll Enable (SPE) bus command by sending an 8-bit status byte when addressed to talk. Bit 6 will be true, logic 1, if the Test Set has sent the SRQ message	SPE SPD STB MTA
Status Bit	No	The Test Set does not have the capability to respond to a Parallel Poll.	PPE PPD PPU PPC IDY

Message Type	Implemented	Response	IEEE 488.1 Message
Clear	Yes	<p>This message clears the Input Buffer and Output Queue, clears any commands in process, puts the Test Set into the Operation Complete idle state and prepares the Test Set to receive new commands. The Device Clear (DCL) or Selected Device Clear (SDC) bus commands</p> <ul style="list-style-type: none"> <li>• do not change any settings or stored data (except as noted previously)</li> <li>• do not interrupt front panel I/O or any Test Set operation in progress (except as noted previously)</li> <li>• do not change the contents of the Status Byte Register (other than clearing the MAV bit as a consequence of clearing the Output Queue).</li> </ul> <p>The Test Set responds equally to DCL or SDC bus commands.</p>	DCL SDC MLA

<b>Message Type</b>	<b>Implemented</b>	<b>Response</b>	<b>IEEE 488.1 Message</b>
Trigger	Yes	If in remote programming mode and addressed to listen, the Test Set makes a triggered measurement following the trigger conditions currently in effect in the instrument. The Test Set responds equally to the Group Execute Trigger (GET) bus command or the *TRG Common Command.	GET MLA
Take Control	Yes	The Test Set begins to act as the Active Controller on the bus.	TCT MTA
Abort	Yes	The Test Set stops talking and listening	IFC

## Remote/Local Modes

### Remote Mode

In Remote mode all front-panel keys are disabled (except for the LOCAL key, POWER switch, Volume control and Squelch control). The LOCAL key is only disabled by the Local Lockout bus command. When in Remote mode and addressed to Listen the Test Set responds to the Data, Remote, Local, Clear (SDC), and Trigger messages. When the Test Set is in Remote mode, the **R** annunciator will be displayed in the upper right corner of the display screen and triggering is set to the state it was last set to in Remote mode (if no previous setting, the default is FULL SETTling and REPetitive RETRiggering). When the Test Set is being addressed to Listen or Talk the **L** or **T** annunciators will be displayed in the upper-right corner of the display screen.

### Local Mode

In Local mode the Test Set's front-panel controls are fully operational. The Test Set uses FULL SETTling and REPetitive RETRiggering in Local mode. When the Test Set is being addressed to Listen or Talk the **L** or **T** annunciators will be displayed in the upper-right corner of the display screen.

### Remote or Local Mode

When addressed to Talk in Remote or Local mode, the Test Set can issue the Data and Status Byte messages and respond to the Take Control message. In addition the Test Set can issue the Service Request Message (SRQ). Regardless of whether it is addressed to talk or listen, the Test Set will respond to the Clear (DCL), Local Lockout, Clear Lockout/Set Local, and Abort messages.

## Local To Remote Transitions

The Test Set switches from Local to Remote mode upon receipt of the Remote message (REN bus line true and Test Set is addressed to listen). No instrument settings are changed by the transition from Local to Remote mode, but triggering is set to the state it was last set to in Remote mode (if no previous setting, the default is FULL SETTling and REPetitive RETRiggering). The R annunciator in the upper-right corner of the display is turned on.

When the Test Set makes a transition from local to remote mode, all currently active measurements are flagged as invalid causing any currently available measurement results to become unavailable. If the HP-IB trigger mode is :RETR REP then a new measurement cycle is started and measurement results will be available for all active measurements when valid results have been obtained. If the HP-IB trigger mode is :RETR SING then a measurement cycle must be started by issuing a trigger event. Refer to "[Triggering Measurements](#)" on page 255 for more information.

## Remote To Local Transitions

The Test Set switches from Remote to Local mode upon receipt of the Local message (Go To Local bus message is sent and Test Set is addressed to listen) or receipt of the Clear Lockout/Set Local message (REN bus line false). No instrument settings are changed by the transition from Remote to Local mode, but triggering is reset to FULL SETTling and REPetitive RETRiggering. The R annunciator in the upper right corner of the display is turned off.

If it is not in Local Lockout mode the Test Set switches from Remote to Local mode whenever the front-panel LOCAL key is pressed.

If the Test Set was in Local Lockout mode when the Local message was received, front-panel control is returned, but Local Lockout mode is not cleared. Unless the Test Set receives the Clear Lockout/Set Local message, the Test Set will still be in Local Lockout mode the next time it goes to the Remote mode.

## Local Lockout

The Local Lockout mode disables the front-panel LOCAL key and allows return to Local mode only by commands from the System Controller (Clear Lockout/Set Local message).

When a data transmission to the Test Set is interrupted, which can happen if the LOCAL key is pressed, the data being transmitted may be lost. This can leave the Test Set in an unknown state. The Local Lockout mode prevents loss of data or system control due to someone unintentionally pressing front-panel keys.

---

### NOTE

Return to Local mode can also be accomplished by setting the POWER switch to OFF and back to ON. However, returning to Local mode in this way has the following disadvantages:

1. It defeats the purpose of the Local Lockout mode in that the Active Controller will lose control of the test set.
  2. Instrument configuration is reset to the power up condition thereby losing the instrument configuration set by the Active Controller.
- 

## Clear Lockout/Set Local

The Test Set returns to Local mode when it receives the Clear Lockout/Set Local message. No instrument settings are changed by the transition from Remote mode with Local Lockout to Local mode but triggering is reset to FULL SETTling and REPetitive RETRiggering.



---

---

**2**

## **Methods For Reading Measurement Results**

## Background

One of the most common remote user interface operations performed on an Test Set is to query and read a measurement result. Generally, this operation is accomplished by sending the query command to the Test Set, followed immediately by a request to read the requested measurement result. Using Hewlett-Packard Rocky Mountain BASIC (RMB) language, this operation would be written using the OUTPUT and ENTER command as follows:

### Query Example

```
OUTPUT 714;"MEAS:RFR:POW?"  
ENTER 714;Power
```

Using this programming structure, the control program will stay on the ENTER statement until it is satisfied - that is - until the Test Set has returned the requested measurement result. This structure works correctly as long as the Test Set returns a valid measurement result. If, for some reason, the Test Set does not return a measurement result, the control program becomes “hung” on the ENTER statement and program execution effectively stops.

In order to prevent the control program from becoming “hung” programmers usually enclose the operation with some form of timeout function. The form of the timeout will of course depend upon the programming language being used. The purpose of the timeout is to specify a fixed amount of time that the control program will wait for the Test Set to return the requested result. After this time has expired the control program will abandon the ENTER statement and try to take some corrective action to regain control of the Test Set.

If the control program does not send the proper commands in the proper sequence when trying to regain control of the Test Set, unexpected operation will result. When this condition is encountered, power must be cycled on the Test Set to regain control.

This situation can be avoided entirely by:

1. sending a Selected Device Clear (SDC) interface message to put the Test Set’s HP-IB subsystem into a known state.
2. sending a command to terminate the requested measurement cycle.

These commands issued in this order will allow the control program to regain control of the Test Set. Any other sequence of commands will result in unexpected operation.

The following programs demonstrate a recommended technique for querying and entering data from the Test Set. This technique will prevent the Test Set from getting into a 'hung' state such that power must be cycled on the Test Set to regain manual or programmatic control.

There are a variety of programming constructs which can be used to implement this technique. In the programming examples presented, a function call is implemented which returns a numeric measurement result. The function call has two pass parameters; the query command (passed as a quoted string) and a time-out value (passed as a integer number).

The time-out value represents how long you want to wait, in seconds, for the Test Set to return a valid measurement result. If a valid measurement result is not returned by the Test Set within the time-out value, the function returns a very large number. The calling program can check the value and take appropriate action.

The program examples are written so as to be self-explanatory. In practice, the length of: variable names, line labels, function names, etc., will be implementation dependent.

## HP BASIC 'ON TIMEOUT' Example Program

The following example program demonstrates a recommended technique which can be utilized in situations where a measurement result timeout value of 32.767 seconds or less is adequate. In the HP RMB language, the timeout parameter for the ON TIMEOUT command has a maximum value of 32.767 seconds. If a timeout value of greater than 32.767 seconds is required refer to the HP BASIC 'MAV' Bit Example Program.

The measurement result timeout value is defined to mean the amount of time the control program is willing to wait for the Test Set to return a valid measurement result to the control program.

Lines 10 thru 230 in this example set up a measurement situation to demonstrate the use of the recommended technique. The recommended technique is exemplified in the Measure Function.

---

### NOTE

Lines 50 and 60 should be included in the beginning of all control programs. These lines are required to ensure that the Test Set is properly reset. This covers the case where the program was previously run and was stopped with the Test Set in an error condition.

---

## Timeout Example

```

10 COM /Io_names/ INTEGER Inst_addr,Bus_addr
20 CLEAR SCREEN
30 Inst_addr=714
40 Bus_addr=7
50 CLEAR Inst_addr
60 OUTPUT Inst_addr;"TRIG:ABORT"
70 OUTPUT Inst_addr;"*RST"
80 OUTPUT Inst_addr;"DISP RFAN"
90 !
100 ! Execute a call to the Measure function with a request to measure RF
110 ! power. The time out value is specified as 10 seconds. The value
120 ! returned by the function is assigned to the variable Measure_result.
130 !
140 Measure_result=FNMeasure("MEAS:RFR:POW?",10)
150 !
160 ! Check the result of the function call.
170 !
180 IF Measure_result=9.E+99 THEN
190   PRINT "Measurement failed."
200 ELSE
210   PRINT "Power = ";Measure_result
220 END IF
230 END
240 !*****
250 ! Recommended Technique:
260 !*****
270 DEF FNMeasure(Query_command$,Time_out_value)
280   COM /Io_names/ INTEGER Inst_addr,Bus_addr
290   DISABLE
300   ON TIMEOUT Bus_addr,Time_out_value RECOVER Timed_out
310   OUTPUT Inst_addr;"TRIG:MODE:RETR SING;:TRIG:IMM"
320   OUTPUT Inst_addr;Query_command$
330   ENTER Inst_addr;Result
340   OUTPUT Inst_addr;"TRIG:MODE:RETR REP"
350   ENABLE
360   RETURN Result
370 Timed_out:!
380   ON TIMEOUT Bus_addr,Time_out_value GOTO Cannot_recover
390   CLEAR Inst_addr
400   OUTPUT Inst_addr;"TRIG:ABORT;MODE:RETR REP"
410   ENABLE
420   RETURN 9.E+99
430 Cannot_recover:!
440   DISP "Cannot regain control of Test Set."
450   STOP
460 FNEEND

```

## Comments for Recommended Routine

Table 2-1

<b>Program Line Number</b>	<b>Comments</b>
50	Send a Selected Device Clear (SDC) to the Test Set to put the HP-IB subsystem into a known state. This allows the control program to regain programmatic control of the Test Set if it is in an error state when the program begins to run.
60	Command the Test Set to abort the currently executing measurement cycle. This will force the Test Set to stop waiting for any measurement results to be available from measurements which may be in an invalid state when the program begins to run.
290	Turn event initiated branches off (except ON END, ON ERROR and ON TIMEOUT) to ensure that the Measure function will not be exited until it is finished.
300	Set up a timeout for any I/O activity on the HP-IB. This will allow the function to recover if the bus hangs for any reason.
310	Set the triggering mode to single followed by a trigger immediate command. This ensures that a new measurement cycle will be started when the TRIG:IMM command is sent. This sequence, that is: set to single trigger and then send a trigger command, guarantees that the measurement result returned to the ENTER statement will accurately reflect the state of the DUT when the TRIG:IMM command was sent. The 'IMM' keyword is optional.
320	Send the query command passed to the Measure function to the Test Set.
330	Read the measurement result.
340	Set the trigger mode to repetitive retriggering. Setting the trigger mode to repetitive will be implementation dependent.
350	Re-enable event initiated branching. If any event initiated branches were logged while the Measure function was executing they will be executed when system priority permits.
360	Exit the Measure function and return the result value.
370	The following lines of code handle the case where the request for a measurement result has timed out.
380	Set up a timeout for any I/O activity on the HP-IB while the control program is trying to regain control of the Test Set. This will allow the function to gracefully stop program execution if the control program cannot regain control of the Test Set. This timeout should only occur if there is some type of hardware failure, either in the Test Set or the external controller, which prevents them from communicating via HP-IB.
390	Send a Selected Device Clear (SDC) to the Test Set to put the HP-IB subsystem into a known state. This allows the control program to regain programmatic control of the Test Set.

<b>Program Line Number</b>	<b>Comments</b>
400	Command the Test Set to abort the currently executing measurement cycle. Set the trigger mode back to repetitive retriggering. Setting the Test Set back to repetitive retriggering will be implementation dependent.
410	Re-enable event initiated branching. If any event initiated branches were logged while the Measure function was executing they will be executed when system priority permits.
420	Exit the Measure function and return a result value of 9.E+99.
430	The following lines of code handle the case where the control program cannot regain control of the Test Set. The actions taken in this section of the code will be implementation dependent. For the example case a message is displayed to the operator and the program is stopped.
440	Display a message to the operator that the control program cannot regain control of the Test Set.
450	Stop execution of the control program.

## HP BASIC 'MAV' Example Program

The following HP RMB example program demonstrates a technique which can be used in situations where a 32.767 measurement result timeout value is not adequate.

Measurement result timeout value is defined to mean the amount of time the control program is willing to wait for the Test Set to return a valid measurement result to the control program.

The technique uses the MAV (Message Available) bit in the Test Set's HP-IB Status Byte to determine when there is data in the Output Queue. A polling loop is used to query the Status byte. The timeout duration for returning the measurement result is handled by the polling loop. An HP-IB interface activity timeout is also set up to handle time-outs resulting from problems with the HP-IB interface.

Lines 10 thru 230 in this example set up a measurement situation to demonstrate the use of the recommended technique. The recommended technique is exemplified in the Measure Function.

---

### NOTE

Lines 50 and 60 should be included in the beginning of all control program. These lines are required to ensure that the Test Set is properly reset. This covers the case where the program was previously run and was stopped with the Test Set in an error condition.

---



## MAV Example

```

10 COM /Io_names/ INTEGER Inst_addr,Bus_addr
20 CLEAR SCREEN
30 Inst_addr=714
40 Bus_addr=7
50 CLEAR Inst_addr
60 OUTPUT Inst_addr;"TRIG:ABORT"
70 OUTPUT Inst_addr;"*RST"
80 OUTPUT Inst_addr;"DISP RFAN"
90 !
100 ! Execute a call to the Measure function with a request to measure RF
110 ! power. The time out value is specified as 50 seconds. The value
120 ! returned by the function is assigned to the variable Measure_result.
130 !
140 Measure_result=FNMeasure("MEAS:RFR:POW?",50)
150 !
160 ! Check the result of the function call.
170 !
180 IF Measure_result=9.E+99 THEN
190 PRINT "Measurement failed."
200 ELSE
210 PRINT "Power = ";Measure_result
220 END IF
230 END
240 !*****
250 ! Recommended Technique:
260 !*****
270 DEF FNMeasure(Query_command$,Time_out_value)
280 COM /Io_names/ INTEGER Inst_addr,Bus_addr
290 DISABLE
300 ON TIMEOUT Bus_addr,5 GOTO Timed_out
310 OUTPUT Inst_addr;"TRIG:MODE:RETR SING::TRIG:IMM"
320 OUTPUT Inst_addr;Query_command$
330 Start_time=TIMEDATE
340 REPEAT
350 WAIT .1
360 Status_byte=SPOLL(Inst_addr)
370 IF BIT(Status_byte,4) THEN
380 ENTER Inst_addr;Result
390 OUTPUT Inst_addr;"TRIG:MODE:RETR REP"
400 ENABLE
410 RETURN Result
420 END IF
430 UNTIL TIMEDATE-Start_time>=Time_out_value
440 Timed_out:!
450 ON TIMEOUT Bus_addr,5 GOTO Cannot_recover
460 CLEAR Inst_addr
470 OUTPUT Inst_addr;"TRIG:ABORT;MODE:RETR REP"
480 RETURN 9.E+99
490 Cannot_recover: !
500 DISP "Cannot regain control of Test Set."
510 STOP
520 FNEEND

```

## Comments for Recommended Routine

**Table 2-2** Comments for Measure Function from MAV Example Program

<b>Program Line Number</b>	<b>Comments</b>
50	Send a Selected Device Clear (SDC) to the Test Set to put the HP-IB subsystem into a known state. This allows the control program to regain programmatic control of the Test Set if it is in an error state when the program begins to run.
60	Command the Test Set to abort the currently executing measurement cycle. This will force the Test Set to stop waiting for any measurement results to be available from measurements which may be in an invalid state when the program begins to run.
290	Turn event initiated branches off (except ON END, ON ERROR and ON TIMEOUT) to ensure that the Measure function will not be exited until it is finished.
300	Set up a 5 second timeout for any I/O activity on the HP-IB. This will allow the function to recover if the bus hangs for any reason. The length of the timeout will be implementation dependent.
310	Set the triggering mode to single followed by a trigger immediate command. This ensures that a new measurement cycle will be started when the TRIG:IMM command is sent. This sequence, that is: set to single trigger and then send trigger command, guarantees that the measurement result returned to the ENTER statement will accurately reflect the state of the DUT when the TRIG:IMM command was sent. The 'IMM' keyword is optional.
320	Send the query command passed to the Measure function to the Test Set.
330	Establish a start time against which to compare the measurement result timeout value passed to the Measure function.
340	Start the status byte polling loop.
350	Allow the Test Set some time (100 milliseconds) to process the measurement. When polling the Test Set the polling loop must give the Test Set time to process the requested measurement. Since HP-IB command processing has a higher system priority within the Test Set than measurement functions, constantly sending HP-IB commands will result in longer measurement times.
360	Perform a serial poll to read the Status Byte from the Test Set. A serial poll is used because the *STB Common Command cannot be processed by the Test Set while a query is pending. Sending the *STB command will cause an 'HP-IB Error: -410 Query INTERRUPTED' error.
370	Check bit 4, the Message Available bit (MAV), to see if it is set to '1'. If it is, then the requested measurement result is ready.
380	Read the measurement result.
390	Set the trigger mode to repetitive retriggering. Setting the trigger mode to repetitive will be implementation dependent.

<b>Program Line Number</b>	<b>Comments</b>
400	Re-enable event initiated branching. If any event initiated branches were logged while the Measure function was executing they will be executed when system priority permits.
410	Exit the Measure function and return the result value.
430	Check to see if the measurement result time out value has been equaled or exceeded. If it has the polling loop will be exited.
440	The following lines of code handle the case where the request for a measurement result has timed out because the polling loop has completed with no result available.
450	Set up a timeout for any I/O activity on the HP-IB while the control program is trying to regain control of the Test Set. This will allow the function to gracefully stop program execution if the control program cannot regain control of the Test Set. This timeout should only occur if there is some type of hardware failure, either in the Test Set or the external controller, which prevents them from communicating via HP-IB.
460	Send a Selected Device Clear (SDC) to the Test Set to put the HP-IB subsystem into a known state. This allows the control program to regain programmatic control of the Test Set.
470	Command the Test Set to abort the currently executing measurement cycle. Set the trigger mode back to repetitive retriggering. Setting the Test Set back to repetitive retriggering will be implementation dependent.
480	Exit the Measure function and return a result value of 9.E+99.
490	The following lines of code handle the case where the control program cannot regain control of the Test Set. The actions taken in this section of the code will be implementation dependent. For the example case a message is displayed to the operator and the program is stopped.
500	Display a message to the operator that the control program cannot regain control of the Test Set.
510	Stop execution of the control program.



---

## **3 HP-IB Command Guidelines**

## Sequential and Overlapped Commands

IEEE 488.2 makes the distinction between sequential and overlapped commands. Sequential commands complete their task before execution of the next command can begin. Overlapped commands can run concurrently, that is, a command following an overlapped command may begin execution while the overlapped command is still in progress. All commands in the Test Set are sequential.

The processing architecture of the Test Set allows it to accept commands through the HP-IB while it is executing commands already parsed into its command buffer. While this may appear to be overlapped, commands are always executed sequentially in the order received.

The process of executing a command can be divided into three steps:

1. Command is accepted from HP-IB and checked for proper structure and parameters.
2. Commands is sent to instrument hardware.
3. Instrument hardware fully responds after some time,  $\Delta t$ .

For example, in programming the Test Set's RF Signal Generator it takes  $< 150$  ms after receipt of the frequency setting command for the output signal to be within 100 Hz of the desired frequency. In the Test Set, commands are considered to have "completed their task" at the end of step 2. In manual operation all displayed measurement results take into account the instrument hardware's response time. When programming measurements through HP-IB the Triggering mode selected will determine whether the instrument's response time is accounted for automatically or if the control program must account for it. Refer to "[Triggering Measurements](#)" on page 255 for a discussion of the different Trigger modes available in the Test Set and their affect on measurement results.

---

## Guidelines for Operation

The following topics discuss rules and guidelines for controlling the Test Set through HP-IB.

### Command Names

All command names of more than four characters have an alternate abbreviated form using only upper case letters and, in some cases, a single numeral. The commands are not case sensitive. Upper and lower case characters can be used for all commands.

For example, to set the destination of AF Generator 1 to Audio Out, any of the following command strings are valid:

#### Example

```
AFGENERATOR1:DESTINATION 'AUDIO OUT'
```

or

```
afgenerator1:destination 'audio out'
```

or

```
afg1:dest 'audio out'
```

or

```
AFG1:DEST 'AUDIO OUT'
```

or

```
Afg1:Dest 'Audio OUT'
```

## Command Punctuation

---

**NOTE**

### Programming Language Considerations

The punctuation rules for the Test Set's HP-IB commands conform to the IEEE 488.2 standard. It is possible that some programming languages used on external controllers may not accept some of the punctuation requirements. It is therefore necessary that the equivalent form of the correct punctuation, as defined by the language, be used for HP-IB operation. Improper punctuation will result in

```
.HP-IB Error: -102 Syntax Error
```

---

### Using Quotes for String Entries

Quotation marks ' and " are used to select a non-numeric field setting. The value is entered into the command line as a quoted alphanumeric string.

Quotes are used with all Underlined (toggling) and One-of-many (menu choice) fields.

For example, to set the RF Generator's Output Port field to Dupl (duplex), the Dupl would be entered into the command string.

#### Example

```
RFG:OUTP 'Dupl'
```

or

```
RFG:OUTP "Dupl"
```

### Using Spaces

When changing a field's setting, a space must always precede the setting value in the command string, regardless of the field type (command<space>value).

#### Example

```
RFG:FREQ<space>850MHZ
```

```
RFG:ATT<space>'OFF'
```

### Using Colons to Separate Commands

The HP-IB command syntax is structured using a control hierarchy that is analogous to manual operation.

The control hierarchy for making a manual instrument setting using the front-panel controls is as follows: first the screen is accessed, then the desired field is selected, then the appropriate setting is made. HP-IB commands use the same hierarchy. The colon (:) is used to separate the different levels of the command hierarchy.



For example, to set the AF Analyzer input gain to 40 dB, the following command syntax would be used:

#### Example

```
DISP AFAN
AFAN:INP:GAIN '40 dB'
```

### Using the Semicolon to Output Multiple Commands

Multiple commands can be output from one program line by separating the commands with a semicolon (;). The semicolon tells the Test Set's HP-IB command parser to back up one level of hierarchy and accept the next command at the same level as the previous command.

For example, on one command line, it is possible to

1. access the AF ANALYZER screen,
2. set the AF Analyzer's Input to AM Demod
3. set Filter 1 to 300 Hz HPF
4. set Filter 2 to 3kHz LPF

#### Example

```
DISP AFAN;AFAN:INP 'AM DEMOD';FILT1 '300Hz HPF';FILT2 '3kHz LPF'
```

The semicolon after the "DISP AFAN" command tells the Test Set's HP-IB command parser that the next command is at the same level in the command hierarchy as the display command. Similarly, the semicolon after the INP 'AM DEMOD' command tells the command parser that the next command (FILT1 '300Hz HPF') is at the same command level as the INP 'AM DEMOD' command.

### Using the Semicolon and Colon to Output Multiple Commands

A semicolon followed by a colon (;:) tells the HP-IB command parser that the next command is at the top level of the command hierarchy. This allows commands from different instruments to be output on one command line. The following example sets the RF Analyzer's tune frequency to 850 MHz, and then sets the AF Analyzer's input to FM Demod.

#### Example

```
RFAN:FREQ 850MHZ;:AFAN:INP 'FM DEMOD'
```

### Using Question Marks to Query Setting or Measurement Fields

The question mark (?) is used to query (read-back) an instrument setting or measurement value. To generate the query form of a command, place the question mark immediately after the command.

Queried information must be read into the proper variable type within the program context before it can be displayed, printed, or used as a numeric value in the program.

Queried information is returned in the same format used to set the value: queried numeric fields return numeric data; quoted string fields return quoted string information.

For example, the following BASIC language program statements query the current setting of the AFGen 1 To field:

#### Example

```
!Query the AFGen1 To field
OUTPUT 714;"AFG1:DEST?"
!Enter queried value into a string variable.
ENTER 714;Afg1_to$
```

## Specifying Units-of-Measure for Settings and Measurement Results

Numeric settings and measurement results in the Test Set can be displayed using one or more units-of-measure (V, mV, Hz, kHz, MHz...). When operating the Test Set manually, the units-of-measure can be easily changed to display measurement results and field settings in the most convenient format. HP-IB operation is similar to manual operation in that the units-of-measure used to display numeric data can be programmatically changed to the most convenient form.

---

#### NOTE

When querying measurements or settings through HP-IB, the Test Set always returns numeric values in HP-IB Units or Attribute Units, regardless of the current Display Units setting. Refer to "[HP-IB Units \(UNITS\)](#)" on page 68 and "[Attribute Units \(AUNits\)](#)" on page 70 for further information.

---

There are three sets of units-of-measure used in the Test Set: Display Units, HP-IB Units, and Attribute Units. Writing correct HP-IB programs requires an understanding of how the Test Set deals with these different sets of units-of-measure.

## Display Units (DUNits)

Display Units are the units-of-measure used by the Test Set to display numeric data (field settings and measurement results) on the front-panel *CRT display*. For example, the RF Generator's frequency can be displayed in Hz, kHz, MHz and GHz. Similarly, the measured TX Frequency can be displayed in Hz, kHz, MHz and GHz.

When evaluating an entered value for a numeric field, the Test Set interprets the data it receives in terms of the Display Units currently set. For example, if the Display Units for the RF Gen Freq field are set to GHz and the operator tries to enter 500 into the field, an Input value out of range error is generated since the Test Set interpreted the value as 500 GHz which is outside the valid frequency range of the Test Set.

**Changing Display Units.** . Use the DUNits command to change the units-of-measure used by the Test Set to display any field setting or measurement result. For example, to change the Display Units setting for the TX Power measurement field from W to dBm, the following command would be used:

### Example

```
MEAS:RFR:POW:DUN DBM
```

### Display Units

```
DUNits Command Example
```

#### GHz

```
:MEAS:RFR:FREQ:ABS:DUN GHZ
```

#### MHz

```
:MEAS:RFR:FREQ:ABS:DUN MHZ
```

#### kHz

```
:MEAS:RFR:FREQ:ABS:DUN KHZ
```

#### Hz

```
:MEAS:RFR:FREQ:ABS:DUN HZ
```

#### ppm

```
:MEAS:RFR:FREQ:ERR:DUN PPM
```

#### %D

```
:MEAS:RFR:FREQ:ERR:DUN PCTDIFF
```

#### V

```
:MEAS:RFR:POW:DUN V
```

**mV**

:MEAS:RFR:POW:DUN MV

**μV**

:RFG:AMPL:DUN μV

**dBμV**

:RFG:AMPL:DUN DBμV

**W**

:MEAS:RFR:POW:DUN W

**mW**

:MEAS:RFR:POW:DUN MW

**dBm**

:MEAS:RFR:POW:DUN DBM

**db**

:MEAS:AFR:DISTN:DUN DB

**%**

:MEAS:AFR:DISTN:DUN PCT

**S**

:DEC:FGEN:GATE:DUN S

**ms**

:DEC:FGEN:GATE:DUN MS

**Reading Back Display Units Setting.** Use the Display Units query command, DUNits?, to read back the current Display Units setting. For example, the following BASIC language program statements query the current Display Units setting for the TX Power measurement:

**Example**

```
!Query Display Units setting for TX Power measurement.  
OUTPUT 714;"MEAS:RFR:POW:DUNits?"  
!Enter the returned value into a string variable.  
ENTER 714;A$
```

The returned units-of-measure will be whatever is shown on the Test Set's front-panel display for the TX Power measurement: dBm, V, mV, dBuV, or W. All returned characters are in upper case. For example, if dBuV is displayed, DBUV is returned.

**Guidelines for Display Units.**

- When querying a field's setting or measurement result through HP-IB, the Test Set always returns numeric values in HP-IB Units or Attribute Units, regardless of the field's current Display Units setting.
- The Display Units for a field's setting or measurement result can be set to any valid unit-of-measure, regardless of the field's HP-IB Units or Attribute Units.
- The Display Units setting for a field's setting is not affected when changing the field's value through HP-IB.

For example, if the AFGen1 Freq Display Units are set to kHz, and the command AFG1:FREQ 10 HZ is sent to change AFGen1's frequency to 10 Hz, the Test Set displays 0.0100 kHz; not 10 Hz.

### HP-IB Units (UNITs)

HP-IB Units are the units-of-measure used by the Test Set when sending numeric data (field settings and measurement results) through HP-IB, and the default units-of-measure for receiving numeric data (field settings and measurement results) through HP-IB. Changing HP-IB Units has no affect on the Display Units or Attribute Units settings. [Table 3-1](#) lists the HP-IB Units used in the Test Set.

**Table 3-1**

**HP-IB Units**

Parameter	Unit of Measure
Power	Watts (W) or dBm (DBM)
Amplitude	Volts (V), or dB $\nu$ V (DBUV)
Frequency	Hertz (Hz)
Frequency Error	Hertz (HZ) or parts per million (PPM)
Time	Seconds (S)
Data Rate	Bits per second (BPS)
Current	Amperes (A)
Resistance	Ohms (OHM)
Relative Level	decibels (DB) or percent (PCT)
Marker Position	Division (DIV)
FM Modulation	Hertz (HZ)
AM Modulation	Percent (PCT)

Use the UNITs? command to determine the HP-IB Units for a measurement result or field setting (refer to "[Reading-Back HP-IB Units](#)" on [page 69](#) for more information).

**Changing HP-IB Units.** Use the UNITS command to change the HP-IB Units setting for selected measurement or instrument setup fields. Only the HP-IB units for power, relative level, and frequency error can be changed. [Table 3-2](#) lists the measurement and instrument setup fields which have changeable HP-IB Units.

**Table 3-2 HP-IB Units That Can Be Changed**

Function	Available HP-IB Units
TX Power measurement	W or DBM
Adjacent Channel Power LRATio, URATio LLEVel, ULEVel	DB or PCT W or DBM
SINAD measurement	DB or PCT
DISTN measurement	DB or PCT
SNR measurement	DB or PCT
RF Generator Amplitude	W or DBM or V or DBUV
Frequency Error	HZ or PPM

For example, the following BASIC language program statements change the HP-IB Units for the TX Power measurement from W to dBm:

**Example**

```
OUTPUT 714;"MEAS:RFR:POW:UNIT DBM"
```

**Reading-Back HP-IB Units.** Use the UNITS? command to read back the current HP-IB Units setting for a measurement or instrument setup field. For example, the following BASIC language program statements read back the current HP-IB Units setting for the TX Power measurement:

**Example**

```
!Query the current HP-IB Units setting for TX Power.
OUTPUT 714;"MEAS:RFR:POW:UNIT?"
!Enter the returned value into a string variable.
ENTER 714;A$
```

### Guidelines for HP-IB Units.

- When setting the value of a numeric field (such as `AFGen1 Freq`), any non-HP-IB Unit unit-of-measure must be specified in the command string, otherwise the current HP-IB Unit is assumed by the Test Set.

For example, if the command `RFG:FREQ 900` is sent through HP-IB, the Test Set will interpret the data as 900 Hz, since HZ is the HP-IB Unit for frequency. This would result in an Input value out of range error. Sending the command `RFG:FREQ 900 MHZ` would set the value to 900 MHz.

- When querying measurements or settings through HP-IB, the Test Set always returns numeric values in HP-IB units, regardless of the current Display Unit setting. Numeric values are expressed in scientific notation.

For example, if the `TX Frequency` measurement is displayed as 150.000000 MHz on the Test Set, the value returned through HP-IB is 1.5000000E+008 ( $1.5 \times 10^8$ ). Converting the returned value to a format other than scientific notation must be done programmatically.

### Attribute Units (AUNits)

Attribute Units are the units-of-measure used by the Test Set when sending or receiving numeric data through HP-IB for the MEASure commands: REFerence, METer (HEND, LEND, INT), HLIMit and LLIMit (refer to "[Number Measurement Syntax](#)" on page 193 for further details). These measurement commands are analogous to the front-panel Data Function keys: REF SET, METER, HI LIMIT and LO LIMIT respectively. Attribute Units use the same set of units-of-measure as the HP-IB Units (except Frequency Error), but are only used with the MEASure commands: REFerence, METer (HEND, LEND, INT), HLIMit and LLIMit. [Table 3-3](#) lists the Attribute Units used in the Test Set.

**Table 3-3**

**Attribute Units**

Parameter	Unit of Measure
Power	Watts (W) or dBm (DBM)
Amplitude	Volts (V)
Frequency	Hertz (Hz)
Time	Seconds (S)
Data Rate	Bits per second (BPS)
Current	Amperes (A)
Resistance	Ohms (OHM)



Parameter	Unit of Measure
Relative Level	decibels (DB) or percent (PCT)
Marker Position	Division (DIV)
FM Modulation	Hertz (HZ)
AM Modulation	Percent (PCT)

**Default Data Function Values.** The majority of measurements made with the Test Set can be made using the Data Functions: REF SET, METER, AVG, HI LIMIT and LO LIMIT. Measurements which can be made using the Data Functions have a cross-reference with the comment “See Number Measurement Syntax” in their syntax. If one or more of the Data Functions are not available to that measurement, the Data Function(s) not available will be listed in the syntax.

For each measurement that can be made using the Data Functions, there is a default set of values for each Data Function for that measurement.

For example, the Audio Frequency Analyzer Distortion measurement can be made using all of the Data Functions. This would include REF SET, METER, AVG, HI LIMIT and LO LIMIT. A complete listing of the Distortion measurement’s Data Functions and their default values would appear as follows:

- The Attribute units are: PCT
- The number of Averages is: 10
- The Average state is: 0
- The Reference value is: 1
- The Reference Display units are: PCT
- The Reference state is: 0
- The High Limit is: 0
- The High Limit Display units are: PCT
- The High Limit state is: 0
- The Low Limit is: 0
- The Low Limit Display units are: PCT
- The Low Limit state is: 0
- The Meter state is: 0
- The Meter high end setting is: 10
- The Meter high end Display units are: PCT

- The Meter low end setting is: 0
- The Meter low end Display units are: PCT
- The Meter interval is: 10

The Data Functions are set to their default values whenever

- the power is cycled on the Test Set
- the front-panel PRESET key is selected
- the \*RST Common Command is received through HP-IB

**Changing Attribute Units.** The AUNits command can be used to change the Attribute Units setting for selected measurements. Only the Attribute Units for power and relative level measurements can be changed. [Table 3-4](#) lists the measurements which have changeable Attribute Units.

**Table 3-4** Measurements with Attribute Units That Can Be Changed

Function	Available Attribute Units
TX Power measurement	W or DBM
Adjacent Channel Power	
LRATio, URATio	DB or PCT
LLEVel, ULEVel	W or DBM
SINAD measurement	DB or PCT
DISTN measurement	DB or PCT
SNR measurement	DB or PCT

Before changing the Attribute Units for a selected measurement, the Test Set verifies that all Data Function values can be properly converted from the current unit-of-measure to the new unit-of-measure. The following Data Function settings are checked:

- the Reference value
- the High Limit
- the Low Limit
- the Meter's high end setting
- the Meter's low end setting
- the Meter's interval

If it is not possible to properly convert all the values to the new unit-of-measure, the Attribute Units are not changed and the following error is generated: HP-IB Error: HP-IB Units cause invalid conversion of attr. This error is most often encountered when one of the Data Function values listed above is set to zero. If this error is encountered, the programmer must change the Data Function settings to values that can be converted to the new units-of-measure before sending the :AUNits command to the Test Set.

For example, the following BASIC language program statements

1. reset the Test Set
2. set the Data Function default zero values to non-zero values
3. set the Attribute Units to DB
4. then query the value of each Data Function

The units of measure for the returned values will be DB.

Display Units and HP-IB Units are not affected when changing Attribute Units.

### Example

```
!Reset the Test Set
OUTPUT 714;"*RST"
!Set HIGH LIMIT value to 15
OUTPUT 714;"MEAS:AFR:DIST:HLIM:VAL 15"
!Set LOW LIMIT value to 1
OUTPUT 714;"MEAS:AFR:DIST:LLIM:VAL 1"
!Set the Meter Lo End value to 1
OUTPUT 714;"MEAS:AFR:DIST:MET:LEND 1"
!Set Attribute Units for Distortion measurement to DB
OUTPUT 714;"MEAS:AFR:DIST:AUN DB"
!Query the REFERENCE SET value
OUTPUT 714;"MEAS:AFR:DIST:REF:VAL?"
!Read the REFERENCE SET value into variable Ref_set_val
ENTER 714;Ref_set_val
!Query the HIGH LIMIT value
OUTPUT 714;"MEAS:AFR:DIST:HLIM:VAL?"
!Read the HIGH LIMIT value into variable Hi_limit_val
ENTER 714;Hi_limit_val
!Query the LOW LIMIT value
OUTPUT 714;"MEAS:AFR:DIST:LLIM:VAL?"
!Read the LOW LIMIT value into variable Lo_limit_val
ENTER 714;Lo_limit_val
!Query the Meter Hi End value
```

```
OUTPUT 714;"MEAS:AFR:DIST:MET:HEND?"
!Read the Meter Hi End value into variable Met_hiend_val
ENTER 714;Met_hiend_val
!Query the Meter Lo End value
OUTPUT 714;"MEAS:AFR:DIST:MET:LEND?"
!Read the Meter Lo End value into variable Met_loend_val
ENTER 714;Met_loend_val
!Query the Meter interval
OUTPUT 714;"MEAS:AFR:DIST:MET:INT?"
!Read the Meter interval into! variable Met_int_val
ENTER 714;Met_int_val
```

### Reading-back Attribute Units.

Use the AUNits? command to read back the Attribute Units setting for the selected measurement. For example, the following BASIC language program statements show how the AUNits? command can be used to read-back a Distortion REFERENCE SET level:

#### Example

```
!Query the REFERENCE SET value for the Distortion measurement
OUTPUT 714;"MEAS:AFR:DIST:REF:VAL?"
!Read the REFERENCE SET value into variable Ref_set_val
ENTER 714;Ref_set_val
!Query the Attribute Units setting for the Distortion measurement
OUTPUT 714;"MEAS:AFR:DIST:AUN?"
!Read the Attribute Units setting into string variable Attribute_set$
ENTER 714;Attribute_set$
!Print out the variables in the form <VALUE><UNITS>
PRINT Ref_set_val;Attribute_set$
```

If a reference of 25% is set, 25 PCT would be printed.

### Guidelines for Attribute Units.

- When setting the value of measurement functions REFERENCE, METer, HLIMit and LLIMit through HP-IB, a non-Attribute Unit unit-of-measure must be specified in the command string, otherwise the current Attribute Unit is assumed by the Test Set.

For example, if the Test Set is in a RESET condition and the command MEAS:AFR:DIST:REF:VAL 10 is sent through HP-IB, the Test Set will interpret the data as 10 %, since % is the RESET Attribute Unit for the Distortion measurement. Sending the command, MEAS:AFR:DIST:REF:VAL 10 DBM, would set the REFERENCE SET value to 10 dB.

- When querying measurement functions REFERENCE, METER, HLIMIT and LLIMIT through HP-IB, the Test Set always returns numeric values in Attribute Units, regardless of the current Display Units or HP-IB Units settings. Numeric values are expressed in scientific notation.

For example, if the REF SET measurement function is displayed as 25% on the Test Set, the value returned through HP-IB is +2.50000000E+001 ( $2.5 \times 10^1$ ). Converting the returned value to a format other than scientific notation must be done programmatically.

- Before changing the Attribute Units for a selected measurement, the Test Set verifies that all Data Function values can be properly converted from the current unit-of-measure to the new unit-of-measure. If it is not possible to properly convert all the values to the new unit-of-measure, the Attribute Units are not changed and the following error is generated: HP-IB Error: HP-IB Units cause invalid conversion of attr.

## Using the STATE Command

The STATE command corresponds to the front-panel ON/OFF key and is used to programmatically turn measurements, instrument functions, and data functions ON or OFF.

### Turning measurements, instrument functions and data functions ON/OFF

Use 1 or ON to turn measurements, instrument functions, or data functions ON. Use 0 or OFF to turn measurements, instrument functions, or data functions OFF.

For example, the following BASIC language statements illustrate the use of the STATE command to turn several measurements, instrument functions, and data functions ON and OFF:

#### Example

```
!Turn off FM source AFG1. *
OUTPUT 714;"AFG1:FM:STAT OFF"

!Turn off REFERENCE SET data function
OUTPUT 714;"MEAS:AFR:DISTN:REF:STAT OFF"

!Turn off TX Power measurement
OUTPUT 714;"MEAS:RFR:POW:STAT 0"

!Turn on REF SET measurement function for FM Deviation measurement
OUTPUT 714;"MEAS:AFR:FM:REF:STAT ON"
```

\*This assumes the AFGen1 To field is set to FM.

## Reading back the measurement, instrument function, or data function state

Use the query form of the command, `STAtE?`, to determine the current state of a measurement, instrument function or data function. If a measurement, instrument function, or data function is queried, the returned value will be either a "1" (ON) or a "0" (OFF).

For example, the following BASIC language statements illustrate the use of the `STAtE?` command to determine the current state of the TX Power measurement:

### Example

```
!Query the state of the TX Power measurement
OUTPUT 714;"MEAS:RFR:POW:STAT?"
ENTER 714;State_on_off
IF State_on_off = 1 THEN DISP "TX Power Measurement is ON"
IF State_on_off = 0 THEN DISP "TX Power Measurement is OFF"
```

## STAtE Command Guidelines

- Measurements that are displayed as numbers, or as analog meters using the `METER` function, can be turned on and off.
- The data functions `REference`, `METer`, `HLIMit`, and `LLIMit` can be turned on and off.
- Any instrument function that generates a signal can be turned on and off. This includes the RF Generator, Tracking Generator, AF Generator 1, AF Generator 2, and the Signaling Encoder.
- The Oscilloscope's trace cannot be turned off.
- The Spectrum Analyzer's trace cannot be turned off.

## Sample HP-IB Program

The following program was written on an HP 9000 Series 300 controller using Hewlett-Packard Rocky Mountain BASIC (RMB). To run this program directly in the Test Set's IBASIC Controller make the following modifications:

1. Use exclamation marks (!) to comment-out lines 440, 450, and 460 (these commands not supported in IBASIC).
2. Change line 70 to `Bus = 8` (internal HP-IB select code = 8).

### Example

```
10 ! This program generates an FM carrier, measures and displays the
20 !deviation, and draws the modulation waveform from the
30 !oscilloscope to the CRT display. For demonstration purposes the
```

```

40 ! carrier is generated and analyzed through the uncalibrated input
50! path so that no external cables are required.
60 GCLEAR !Clear graphics display.
70 Bus=7 ! Interface select code of HP-IB interface
80 Dut=100*Bus+14 ! Default Test Set HP-IB address is 14
90 CLEAR Bus! Good practice to clear the bus
100 CLEAR SCREEN ! Clear the CRT
110 OUTPUT Dut;"*RST" ! Preset the Test Set
120 OUTPUT Dut;"DISP DUPL" ! Display the DUPLEX TEST screen
130 OUTPUT Dut;"RFG:AMPL -14 DBM" ! Set RF Gen Amptd to -14 dBm
140 OUTPUT Dut;"AFAN:INP 'FM Demod'"
150 ! Set AF Analyzer's input to FM Demod
160 OUTPUT Dut;"AFAN:DET 'Pk+/-Max'"
170 ! Set AF Analyzer's detector to Peak +/-Max
180 ! The following trigger guarantees the instrument will auto-tune
190 !and auto-range to the input signal before measuring.
200 OUTPUT Dut;"TRIG"! Trigger all active measurements
210 OUTPUT Dut;"MEAS:AFR:FM?" ! Request an FM deviation measurement
220 ENTER Dut;Dev ! Read measured value into variable Dev
230 PRINT USING "K,D.DDD,K";"Measured FM = ",Dev/1000," kHz peak."
240 DISP "'Continue' when ready..." ! Set up user prompt
245 ! Set up interrupt on softkey 1
250 ON KEY 1 LABEL "Continue",15 GOTO Proceed
260 LOOP! Loop until the key is pressed
270 END LOOP
280 Proceed: OFF KEY! Turn off interrupt from softkey 1
290 DISP "! Clear the user prompt
300 !
310 !Measure and plot oscilloscope trace to see the waveform shape.
320 DIM Trace(0:416)! Oscilloscope has 417 trace points
330 OUTPUT Dut;"DISP OSC" Display the Oscilloscope screen
340 OUTPUT Dut;"TRIG"! Trigger all active measurements
350 OUTPUT Dut;"MEAS:OSC:TRAC?"
360 !Request the oscilloscope trace
370 ENTER Dut;Trace(*)
380 ! Read the oscilloscope trace into array Trace(*)
390 ! CRT is (X,Y)=(0,0) in lower left corner
400 !to (399,179) upper right.
410 ! (Each pixel is about 0.02 mm wide by 0.03 mm tall, not square.)
420 ! Scale vertically for 0 kHz dev center-screen and +4 kHz dev top
430 ! of screen. Leave the next three lines for external control, or

```

```
440 ! comment them out for IBASIC (Test Set stand-alone) control.
450 !
460 PLOTTER IS CRT,"98627A"
470 !Your display may have a different specifier.
480 GRAPHICS ON!Enable graphics to plot the waveform.
490 WINDOW 0,399,0,179
500 !
510 PEN 1 !Turn on drawing pen
520 MOVE 0,89.5+Trace(0)/4000*89.5
530 FOR I=1 TO 416
540 DRAW I/416*399,89.5+Trace(I)/4000*89.5
550 NEXT I
560 END
```



---

## **4 HP-IB Command List**

<b>:AFANalyzer subsystem</b> .....	<b>112</b>
:AIN <string>	
:AIN? .....	112
:DEMPHasis <string>	
:DEMPHasis? .....	112
:DEMPHasis:GAIN <string>	
:DEMPHasis:GAIN? .....	113
:DETEctor <string>	
:DETEctor? .....	113
:DETEctor:PKLocation <string>	
:DETEctor:PKLocation? .....	114
:DETEctor:SETTling <string>	
:DETEctor:SETTling? .....	114
:ELResistor <real number>	
:ELResistor? .....	114
:FILT1   FILTER1 <string>	
:FILT1?   FILTER1? .....	115
:FILT2   FILTER2 <string>	
:FILT2?   FILTER2? .....	115
:GTIME <real number>	
:GTIME? .....	116
:INPut <string>	
:INPut? .....	116
:INPut:GAIN <string>	
:INPut:GAIN? .....	116
:SMPoint <string>	
:SMPoint? .....	117
:NOTCh:GAIN <string>	
:NOTCh:GAIN? .....	118
:NOTCh:FREQuency <real number>	
:NOTCh:FREQuency? .....	118
:RANGing <string>	
:RANGing? .....	118
:SPEaker:MODE <string>	
:SPEaker:MODE? .....	119
:SPEaker:VOLume <string>	
:SPEaker:VOLume? .....	119
<b>:AFG1   AFGENERATOR1 subsystem</b> .....	<b>120</b>
:AM <real number>	
:AM? .....	120
:DESTination <string>	
:DESTination? .....	120
:FM <real number>	
:FM? .....	120
:FREQuency <real number>	
:FREQuency? .....	121
:OUTPut <real number>	
:OUTPut? .....	121
<b>:AFG2   AFGENERATOR2 subsystem</b> .....	<b>122</b>

:AM <real number>	
:AM? .....	122
:DESTination <string>	
:DESTination? .....	122
:FM <real number>	
:FM? .....	122
:FREQuency <real number>	
:FREQuency? .....	123
:OUTPut <real number>	
:OUTPut? .....	123
<b>:CANalyzer subsystem .....</b>	<b>124</b>
:ARM .....	124
:AUTO:GAIN? .....	124
:AUTO:POWer:GAIN? .....	124
:CHANnel:POWer:CALibrate .....	125
:DARM .....	125
:DIRection <string>	
:DIRection? .....	126
:EVENT:QUALifier <string>	
:EVENT:QUALifier? .....	126
:EVENT:TRIGger <string>	
:EVENT:TRIGger? .....	127
:EVENT:TRIGger:DELay <real number>	
:EVENT:TRIGger:DELay? .....	127
:MODE <string>	
:MODE? .....	128
:PATH:GAIN <string>	
:PATH:GAIN? .....	128
:PATH:GAIN:MODE <string>	
:PATH:GAIN:MODE? .....	129
:PNOffset <real number>	
:PNOffset? .....	129
:POWer:GAIN <string>	
:POWer:GAIN? .....	130
:POWer:GAIN:MODE <string>	
:POWer:GAIN:MODE? .....	130
:POWer:SAMple:TIME <real number>	
:POWer:SAMple:TIME? .....	131
:POWer:ZERO .....	131
:POWer:ZERO:MODE <string>	
:POWer:ZERO:MODE? .....	132
:SAMple:TIME <real number>	
:SAMple:TIME? .....	132
:SPECial <string>	
:SPECial? .....	132
:TRIGger:STATE	
:TRIGger:STATE? .....	133
<b>:CBUffer subsystem .....</b>	<b>134</b>
:DATA <48 or 72 character hex string>	

:DATA?	134
:FRAME:COUNT <integer>,<real number>	
:FRAME:COUNT?	134
:FRAME:START <integer>,<real number>	
:FRAME:START?	135
:FRAME:LOAD <integer>,<real number>	
:FRAME:LOAD?	135
:MODE <string>	
:MODE?	135
:STATE <string>	
:STATE?	136
<b>:CCOMmon subsystem</b>	<b>137</b>
:PATH <string>	
:PATH?	137
<b>:CDANalyzer subsystem</b>	<b>138</b>
:CONTrol <string>	
:CONTrol?	138
:EVENT:QUALifier <string>	
:EVENT:QUALifier?	138
:EVENT:TRIGger <string>	
:EVENT:TRIGger?	139
:EVENT:TRIGger:DELay <real number>	
:EVENT:TRIGger:DELay?	139
:MARKer:POSition <real number>	
:MARKer:POSition?	140
:MEASure <string>	
:MEASure?	140
:FPOWER:NAVG <integer>	
:FPOWER:NAVG?	140
:FPOWER:TOFFset <real number>	
:FPOWER:TOFFset?	141
:FPOWER:TOFFset:TRANsfer	141
:MODE <string>	
:MODE?	141
:PATH:GAIN <real number>	
:PATH:GAIN?	142
:POWER:REFerence <string>	
:POWER:REFerence?	142
:SAMPLe:TIME <real number>	
:SAMPLe:TIME?	142
:SCALE:PHASe <string>	
:SCALE:PHASe?	142
:SCALE:POWER <string>	
:SCALE:POWER?	143
:SCALE:TIME <string>	
:SCALE:TIME?	144
:THRShld <real number>	
:THRShld?	144
:TRIGger:ARM	

:TRIGger:DARM .....	145
<b>:CGENERator subsystem .....</b>	<b>146</b>
:CONTRol <string>	
:CONTRol? .....	146
:DATA:SOURce <string>	
:DATA:SOURce? .....	146
:DATA:RATE <string>	
:DATA:RATE? .....	146
:DIRection <string>	
DIRection? .....	147
:EBNO:LEVel <real number>	
:EBNO:LEVel? .....	147
:EQFilter <string>	
:EQFilter? .....	148
:EVENsec <string>	
:EVENsec? .....	148
:SPECial <string>	
:SPECial? .....	148
<b>:CONFigure subsystem .....</b>	<b>149</b>
:SRLocation <string>	
:SRLocation? .....	149
:BADDRESS <integer> .....	149
:BEEPer <string>	
:BEEPer? .....	150
:DISPlay:MESSages	
:DISPlay:MESSages? .....	150
:BMODE <string>	
:BMODE? .....	150
:DATE <integer>	
:DATE? .....	151
:FCLK <string>	
:FCLK? .....	151
:RFDisplay <string>	
:RFDisplay? .....	151
:KNOB <string>	
:KNOB? .....	152
:NOTChmode <string>	
:NOTChmode? .....	152
:OFLevel:MODE <string>	
:OFLevel:MODE? .....	152
:OFLevel:ANTenna <real number>	
:OFLevel:ANTenna? .....	153
:OFLevel:DUplex <real number> .....	153
:OFLevel:RFINout <real number> .....	154
:OFRequency <real number> .....	154
:OMODE <string>	
:OMODE? .....	154
:OPERation:AUTO	
:OPERation:HOLD .....	155

:PRINt:ADDDress <integer>	
:PRINt:ADDDress? .....	155
:PRINt:LINEs   LINE <integer>	
:PRINt:LINEs   LINE? .....	155
:PRINt:DEStination   PORTs <string>	
:PRINt:DEStination?   PORTs? .....	156
:PRINt:FFStArt <string>	
:PRINt:FFStArt? .....	156
:PRINt:FFENd <string>	
:PRINt:FFENd? .....	156
:PRINt:TITLe <string>	
:PRINt:TITLe? .....	157
:PRINt:PRINter   :PRINt:HPModel   :PRINt:HPMO <string>	
:PRINt:PRINter?   :PRINt:HPModel?   :PRINt:HPMO? .....	157
:REFeRence:INPut:SElect <string>	
:REFeRence:INPut:SElect .....	158
:REFeRence:INPut:EXtErnal <string>	
:REFeRence:INPut:EXtErnal? .....	158
:RFIMped <string>	
:RFIMped? .....	159
:SPSelect <string>	
:SPSelect? .....	159
:SPOR9:BAUD   SPORT9:BAUD <string>	
:SPOR9:BAUD?   SPORT9:BAUD? .....	160
:SPOR9:PARity   SPORT9:PARity <string>	
:SPOR9:PARity?   SPORT9:PARity? .....	160
:SPOR9:DATA   SPORT9:DATA <string>	
:SPOR9:DATA?   SPORT9:DATA? .....	160
:SPOR9:StOP   SPORT9:StOP <string>	
:SPOR9:StOP?   SPORT9:StOP? .....	161
:SPOR9:FCONtrol   SPORT9:FCONtrol <string>	
:SPOR9:FCONtrol?   SPORT9:FCONtrol?	
:SPOR9:FLOW   SPORT9:FLOW <string>	
:SPOR9:FLOW?   SPORT9:FLOW? .....	161
:SPOR9:SiN   SPORT9:SiN <string>	
:SPOR9:SiN?   SPORT9:SiN?	
:SPOR9:SiNPut   SPORT9:SiNPut <string>	
:SPOR9:SiNPut?   SPORT9:SiNPut? .....	162
:SPOR9:IBECho   SPORT9:IBECho <string>	
:SPOR9:IBECho?   SPORT9:IBECho? .....	162
:SPOR9:IECHo   SPORT9:IECHo <string>	
:SPOR9:IECHo?   SPORT9:IECHo? .....	162
:SPOR9:StAtus:LINE? .....	163
:SPOR9:StAtus:MODem? .....	163
:SPOR10:BAUD   SPORT10:BAUD <string>	
:SPOR10:BAUD?   SPORT10:BAUD? .....	163
:SPOR10:PARity   SPORT10:PARity <string>	
:SPOR10:PARity?   SPORT10:PARity? .....	164
:SPOR10:DATA   SPORT10:DATA <string>	
:SPOR10:DATA?   SPORT10:DATA? .....	164
:SPOR10:StOP   SPORT10:StOP <string>	
:SPOR10:StOP?   SPORT10:StOP? .....	164
:SPOR10:FCONtrol   SPORT10:FCONtrol <string>	

:SPOR10:FCONtrol?   SPORT10:FCONtrol?	
:SPOR10:FLOW   SPORT10:FLOW <string>	
:SPOR10:FLOW?   SPORT10:FLOW? .....	165
:SPOR10:SIN   SPORT10:SIN <string>	
:SPOR10:SIN?   SPORT10:SIN?	
:SPOR10:SINPut   SPORT10:SINPut <string>	
:SPOR10:SINPut?   SPORT10:SINPut? .....	165
:SPOR10:IBECho   SPORT10:IBECho <string>	
:SPOR10:IBECho?   SPORT10:IBECho? .....	165
:SPOR10:IECHo   SPORT10:IECHo <string>	
:SPOR10:IECHo?   SPORT10:IECHo? .....	167
:SPOR10:STATus:LINE? .....	167
:SPOR10:STATus:MODem? .....	167
:SPOR11:BAUD   SPORT11:BAUD <string>	
:SPOR11:BAUD?   SPORT11:BAUD? .....	167
:SPOR11:PARity   SPORT11:PARity <string>	
:SPOR11:PARity?   SPORT11:PARity? .....	168
:SPOR11:DATA   SPORT11:DATA <string>	
:SPOR11:DATA?   SPORT11:DATA? .....	168
:SPOR11:STOP   SPORT11:STOP <string>	
:SPOR11:STOP?   SPORT11:STOP? .....	169
:SPOR11:FCONtrol   SPORT11:FCONtrol <string>	
:SPOR11:FCONtrol?   SPORT11:FCONtrol?	
:SPOR11:FLOW   SPORT11:FLOW <string>	
:SPOR11:FLOW?   SPORT11:FLOW? .....	169
:SPOR11:SIN   SPORT11:SIN <string>	
:SPOR11:SIN?   SPORT11:SIN?	
:SPOR11:SINPut   SPORT11:SINPut <string>	
:SPOR11:SINPut?   SPORT11:SINPut? .....	169
:SPOR11:IBECho   SPORT11:IBECho <string>	
:SPOR11:IBECho?   SPORT11:IBECho? .....	170
:SPOR11:IECHo   SPORT11:IECHo <string>	
:SPOR11:IECHo?   SPORT11:IECHo? .....	170
:SPOR11:STATus:LINE? .....	170
:SPOR11:STATus:MODem? .....	171
:TBASE:CDMA <string>	
:TBASE:CDMA? .....	171
:TIME <real number>	
:TIME? .....	172
<b>:DISPlay subsystem .....</b>	<b>173</b>
:DISPlay	
:DISPlay? .....	173
<b>Integer Number Setting Syntax .....</b>	<b>175</b>
Syntax .....	175
<b>:MEASure subsystem .....</b>	<b>176</b>
:RESet .....	176
:AFRequency:ACLevel <measurement command>	

:AFRequency:ACLevel? . . . . .	176
:AFRequency:AM <measurement command>	
:AFRequency:AM? . . . . .	176
:AFRequency:CURRent <measurement command>	
:AFRequency:CURRent? . . . . .	177
:AFRequency:DCAM <measurement command>	
:AFRequency:DCAM? . . . . .	178
:AFRequency:DCFM <measurement command>	
:AFRequency:DCFM? . . . . .	178
:AFRequency:DCVolts <measurement command>	
:AFRequency:DCVolts? . . . . .	179
:AFRequency:DISTortion <measurement command>	
:AFRequency:DISTortion? . . . . .	179
:AFRequency:FM <measurement command>	
:AFRequency:FM? . . . . .	180
:AFRequency:FREQuency <measurement command>	
:AFRequency:FREQuency? . . . . .	180
:AFRequency:SElect <measurement command>	
:AFRequency:SElect? . . . . .	181
:AFRequency:SINAD <measurement command>	
:AFRequency:SINAD? . . . . .	181
:AFRequency:SNR <measurement command>	
:AFRequency:SNR? . . . . .	182
:CANalyzer:ADC:APOWeradc:LEVel <measurement command>	
:CANalyzer:ADC:APOWeradc:LEVel? . . . . .	182
:CANalyzer:ADC:RHOadc:LEVel <measurement command>	
:CANalyzer:ADC:RHOadc:LEVel? . . . . .	182
:CANalyzer:ADC:EVMadc:LEVel <measurement command>	
:CANalyzer:ADC:EVMadc:LEVel? . . . . .	183
:CANalyzer:ADC:TPOWeradc:LEVel <measurement command>	
:CANalyzer:ADC:TPOWeradc:LEVel? . . . . .	184
:CANalyzer:CARrier:FEEDthrough <measurement command>	
:CANalyzer:CARrier:FEEDthrough? . . . . .	184
:CANalyzer:ERRor:FREQuency <measurement command>	
:CANalyzer:ERRor:FREQuency? . . . . .	185
:CANalyzer:ERRor:MAGNitude <measurement command>	
:CANalyzer:ERRor:MAGNitude? . . . . .	185
:CANalyzer:ERRor:PHASe <measurement command>	
:CANalyzer:ERRor:PHASe? . . . . .	186
:CANalyzer:EVM <measurement command>	
:CANalyzer:EVM? . . . . .	186
:CANalyzer:POWER:AVG <measurement command>	
:CANalyzer:POWER:AVG? . . . . .	186
:CANalyzer:POWER:CHANnel <measurement command>	
:CANalyzer:POWER:CHANnel? . . . . .	187
:CANalyzer:RHO <measurement command>	
:CANalyzer:RHO? . . . . .	188
:CANalyzer:TIME:OFFSet <measurement command>	
:CANalyzer:TIME:OFFSet? . . . . .	188
:CDANalyzer:ADC <measurement command>	
:CDANalyzer:ADC? . . . . .	188
:CDANalyzer:CARrier:FEEDthrough <measurement command>	
:CDANalyzer:CARrier:FEEDthrough? . . . . .	189



:CDANalyzer:ERRor:FREQUency <measurement command>	
:CDANalyzer:ERRor:FREQUency? . . . . .	190
:CDANalyzer:MARKer:LEVel <measurement command>	
:CDANalyzer:MARKer:LEVel? . . . . .	190
:CDANalyzer:MARKer:PHASe <measurement command>	
:CDANalyzer:MARKer:PHASe? . . . . .	191
:CDANalyzer:MARKer:TIME <measurement command>	
:CDANalyzer:MARKer:TIME? . . . . .	191
:CDANalyzer:TIME:OFFSet <measurement command>	
:CDANalyzer:TIME:OFFSet? . . . . .	192
:CDANalyzer:TRACe:POWer? . . . . .	192
:CDANalyzer:TRACe:TOFFset? . . . . .	192
:CDANalyzer:TRACe:PERRor? . . . . .	193
:OSCilloscope:MARKer:LEVel:AM <measurement command>	
:OSCilloscope:MARKer:LEVel:AM? . . . . .	194
:OSCilloscope:MARKer:LEVel:FM <measurement command>	
:OSCilloscope:MARKer:LEVel:FM? . . . . .	194
:OSCilloscope:MARKer:LEVel:VOLTs <measurement command>	
:OSCilloscope:MARKer:LEVel:VOLTs? . . . . .	195
:OSCilloscope:MARKer:TIME <measurement command>	
:OSCilloscope:MARKer:TIME? . . . . .	195
:OSCilloscope:TRACe? . . . . .	196
:RFRequency:SElect <string>	
:RFRequency:SElect? . . . . .	196
:RFRequency:FREQUency:ABSolute <measurement command>	
:RFRequency:FREQUency:ABSolute? . . . . .	196
:RFRequency:FREQUency:ERRor <measurement command>	
:RFRequency:FREQUency:ERRor? . . . . .	197
:RFRequency:POWer <measurement command>	
:RFRequency:POWer? . . . . .	197
:SANalyzer:MARKer:FREQUency <measurement command>	
:SANalyzer:MARKer:FREQUency? . . . . .	198
:SANalyzer:MARKer:LEVel <measurement command>	
:SANalyzer:MARKer:LEVel? . . . . .	198
:SANalyzer:TRACe? . . . . .	199
<b>Multiple Real Number Setting Syntax . . . . .</b>	<b>200</b>
Syntax . . . . .	200
<b>Number Measurement Syntax . . . . .</b>	<b>201</b>
Syntax . . . . .	201
<b>:OSCilloscope subsystem . . . . .</b>	<b>204</b>
:CONTRol	
:CONTRol? . . . . .	204
:MARKer:NPEak . . . . .	204
:MARKer:PPEak . . . . .	204
:MARKer:POSition <real number>	
:MARKer:POSition? . . . . .	205
:SCALE:TIME <string>	

:SCALE:TIME?	206
:SCALE:NADC <integer>	
:SCALE:NADC?	206
:SCALE:VERTical:AM <string>	
:SCALE:VERTical:AM?	207
:SCALE:VERTical:FM <string>	
:SCALE:VERTical:FM?	208
:SCALE:VERTical:OFFSet <real number>	
:SCALE:VERTical:OFFSet?	208
:SCALE:VERTical:VOLTs <string>	
:SCALE:VERTical:VOLTs?	210
:TRIGger:LEVel <real number>	
:TRIGger:LEVel?	211
:TRIGger:MODE <string>	
:TRIGger:MODE?	211
:TRIGger:DELay <real number>	
:TRIGger:DELay?	212
:TRIGger:PRETrigger <real number>	
:TRIGger:PRETrigger?	212
:TRIGger:RESet	212
:TRIGger:SENSe <string>	
:TRIGger:SENSe?	213
:TRIGger:SOURce <string>	
:TRIGger:SOURce?	213
:TRIGger:TYPE <string>	
:TRIGger:TYPE?	214
<b>:PROGram subsystem</b>	<b>215</b>
[:SElected]:DEFine <program data>	
[:SElected]:DEFine?	215
[:SElected]:EXECute	215
[:SElected]:STATe <variable>	
[:SElected]:STATe?	215
[:SElected]:NUMBer <variable>(,<nvalues>)	
[:SElected]:NUMBer?	216
[:SElected]:STRing <variable>(,<nvalues>)	
[:SElected]:STRing?	216
[:SElected]:WAIT	
[:SElected]:WAIT?	216
[:SElected]:DELeTe	
[:SElected]:DELeTe:ALL	216
<b>Real Number Setting Syntax</b>	<b>217</b>
Syntax	217
<b>[:REGister] subsystem</b>	<b>219</b>
:CLEar <integer or string>	219
:CLEar:ALL	219
:RECall <integer or string>	220
:RECall:LIST?	220

:SAVE <integer or string> . . . . .	220
:SAVE:LIST? . . . . .	220
<b>:RFAnalyzer subsystem . . . . .</b>	<b>222</b>
:ATTenuator <string>	
:ATTenuator? . . . . .	222
:ATTenuator:MODE <string>	
:ATTenuator:MODE? . . . . .	222
:FREQuency <real number>	
:FREQuency? . . . . .	222
:FREQuency:GTIME <real number> . . . . .	223
:GTIME <real number>	
:GTIME? . . . . .	223
:IFBW <string>	
:IFBW? . . . . .	224
:INPut <string>	
:INPut? . . . . .	224
:PMEasurement:DETECTOR <string>	
:PMEasurement:DETECTOR? . . . . .	224
:PMEasurement:ZERO . . . . .	225
:SENSitivity <string>	
:SENSitivity? . . . . .	225
:SQUelch <string>	
:SQUelch? . . . . .	225
<b>:RFGenerator subsystem . . . . .</b>	<b>226</b>
:ATTenuator <string>	
:ATTenuator? . . . . .	226
:AMPLitude <real number> . . . . .	226
:FM:COUPLing <string>	
:FM:COUPLing? . . . . .	226
:FM:DCZero . . . . .	227
:FREQuency <real number> . . . . .	227
:MODulation:AOUT <string>	
:MODulation:AOUT? . . . . .	228
:MODulation:EXTernal:AM <real number>	
:MODulation:EXTernal:AM? . . . . .	228
:MODulation:EXTernal:DESTination <string>	
:MODulation:EXTernal:DESTination? . . . . .	228
:MODulation:EXTernal:FM <real number>	
:MODulation:EXTernal:FM? . . . . .	229
:OUTPut <string>	
:OUTPut? . . . . .	229
<b>:SANalyzer subsystem . . . . .</b>	<b>230</b>
:ATTenuator <string>	
:ATTenuator? . . . . .	230
:ATTenuator:MODE <string>	
:ATTenuator:MODE? . . . . .	230
:CFRequency <real number>	

:CFRequency? . . . . .	230
:CONTRol <string>	
:CONTRol? . . . . .	231
:DISPlay:SCALe <string>	
:DISPlay:SCALe . . . . .	231
:INPut <string>	
:INPut . . . . .	232
:MARKer:CFRequency . . . . .	232
:MARKer:NPEak . . . . .	232
:MARKer:EXCURsion <integer number>	
:MARKer:EXCURsion? . . . . .	232
:MARKer:NPLEvel <real number>	
:MARKer:NPLEvel? . . . . .	233
:MARKer:PEAK . . . . .	233
:MARKer:POSItion <real number>	
:MARKer:POSItion? . . . . .	234
:MARKer:RLEVel . . . . .	234
:RFGenerator <string>	
:RFGenerator? . . . . .	234
:RLEVel <real number>	
:RLEVel? . . . . .	235
:SPAN <real number>	
:SPAN? . . . . .	235
:TGENERator:AMPLitude <real number>	
:TGENERator:AMPLitude? . . . . .	235
:TGENERator:DESTination <string>	
:TGENERator:DESTination? . . . . .	236
:TGENERator:OFRequency <real number>	
:TGENERator:OFRequency? . . . . .	236
:TGENERator:SWEEp <string>	
:TGENERator:SWEEp? . . . . .	236
:TRACe:MHOLD <string>	
:TRACe:MHOLD? . . . . .	237
:TRACe:NORMALize <string>	
:TRACe:NORMALize? . . . . .	237
:TRACe:SAVE . . . . .	238
<b>:STATus . . . . .</b>	<b>239</b>
:PRESet . . . . .	239
:CALibration:CONDition? . . . . .	239
:CALibration:ENABle	
:CALibration:ENABle? . . . . .	239
:CALibration[:EVENT]?. . . . .	239
:CALibration:NTRansition	
:CALibration:NTRansition? . . . . .	240
:CALibration[:EVENT?]:PTRansition	
:CALibration[:EVENT?]:PTRansition? . . . . .	240
:COMMunicate:CONDition? . . . . .	240
:COMMunicate:ENABle	
:COMMunicate:ENABle? . . . . .	240
:COMMunicate[:EVENT?]:NTRansition	
:COMMunicate[:EVENT?]:NTRansition? . . . . .	240

:COMMunicate[:EVENT?]:PTRansition	
:COMMunicate[:EVENT?]:PTRansition?	241
:HARD1:CONDition?	241
:HARD1:ENABle	
:HARD1:ENABle?	241
:HARD1[:EVENT?]:NTRansition	
:HARD1[:EVENT?]:NTRansition?	241
:HARD1[:EVENT?]:PTRansition	
:HARD1[:EVENT?]:PTRansition?	242
:HARD2:CONDition?	242
:HARD2:ENABle	
:HARD2:ENABle?	242
:HARD2[:EVENT?]:NTRansition	
:HARD2[:EVENT?]:NTRansition?	242
:HARD2[:EVENT?]:PTRansition	
:HARD2[:EVENT?]:PTRansition?	242
:OPERation:CONDition?	243
:OPERation:ENABle	
:OPERation:ENABle?	243
:OPERation[:EVENT?]:NTRansition	
:OPERation[:EVENT?]:NTRansition?	243
:OPERation[:EVENT?]:PTRansition	
:OPERation[:EVENT?]:PTRansition?	243
:CALibrating:CONDition?	243
:CALibrating:ENABle	
:CALibrating:ENABle?	244
:CALibrating[:EVENT?]:NTRansition	
:CALibrating[:EVENT?]:NTRansition?	244
:CALibrating[:EVENT?]:PTRansition	
:CALibrating[:EVENT?]:PTRansition?	244
:QUESTionable:CONDition?	244
:QUESTionable:ENABle	
:QUESTionable:ENABle?	244
:QUESTionable[:EVENT?]:NTRansition	
:QUESTionable[:EVENT?]:NTRansition?	245
:QUESTionable[:EVENT?]:PTRansition	
:QUESTionable[:EVENT?]:PTRansition?	245
:MEASuring:CONDition?	245
:MEASuring:ENABle	
:MEASuring:ENABle?	245
:MEASuring[:EVENT?]:NTRansition	
:MEASuring[:EVENT?]:NTRansition?	245
:MEASuring[:EVENT?]:PTRansition	
:MEASuring[:EVENT?]:PTRansition?	246
:CDMA1:CONDition?	246
:CDMA1:ENABle	
:CDMA1:ENABle?	246
:CDMA1[:EVENT?]:NTRansition	
:CDMA1[:EVENT?]:NTRansition?	246
:CDMA1[:EVENT?]:PTRansition	
:CDMA1[:EVENT?]:PTRansition?	246
:IBASic:CONDition?	247
:IBASic:ENABle	

:IBASic:ENABLE? .....	247
:IBASic[:EVENT?]:NTRansition	
:IBASic[:EVENT?]:NTRansition? .....	247
:IBASic[:EVENT?]:PTRansition	
:IBASic[:EVENT?]:PTRansition? .....	247
<b>:SYSTEM .....</b>	<b>248</b>
[:ERRor?] .....	248
<b>:TESTs Subsystem .....</b>	<b>249</b>
:COMMENT1   COMM1 <character data>	
:COMMENT1?   COMM1? .....	249
:COMMENT2   COMM2	
:COMMENT2?   COMM2? .....	249
:CONFigure   CNFG	
<integer_value>,<character_data>,<character_data> .....	
<integer_value>,<character_data>	
:CONFigure?   CNFG? .....	250
:EXECution:DESTination <string>	
:EXECution:DESTination? .....	250
:EXECution:FAILure <string>	
:EXECution:FAILure? .....	250
:EXECution:HEADING1   HEAD1 <string>	
:EXECution:HEADING1?   HEAD1? .....	250
:EXECution:HEADING2   HEAD2 <string>	
:EXECution:HEADING2?   HEAD2? .....	251
:EXECution:RESults <string>	
:EXECution:RESults? .....	251
:EXECution:RUN <string>	
:EXECution:RUN? .....	251
:FREQuency <integer>,<real>,<character data>,<real>,<character da- ta>,<string>,<string>	
:FREQuency? .....	251
:LIBRARY? .....	252
:PARAmeter[:NUMBER] <integer>,<real>	
:PARAmeter[:NUMBER?] .....	252
:PARAmeter:STRing <string>,<real>	
:PARAmeter:STRing? .....	252
:PROCCedure:AUTOstart <string>	
:PROCCedure:AUTOstart? .....	252
:PROCCedure:LOCation <string>	
:PROCCedure:LOCation? .....	252
:PROCCedure:NAME <character data>	
:PROCCedure:NAME? .....	253
:PROCCedure:RUN .....	253
:PROCCedure:RUNTest .....	253
:SEQNumber[:NUMBER] <integer>,<string>	
:SEQNumber[:NUMBER?] .....	253
:SPEC[:NUMBER] <integer>,<real>,<real>,<string>	
:SPEC[:NUMBER?] .....	254
:SPEC:STRing <real>,<real>,<string>	

:SPEC:STRing? .....	254
<b>:TRIGger subsystem.....</b>	<b>255</b>
:ABORt .....	255
[:IMMediate] .....	255
:MODE:RETRigger	
:MODE:RETRigger?.....	255
:MODE:SETTling	
:MODE:SETTling?.....	255





---

---

# 5

## **HP-IB Command Dictionary**

This is the HP-IB command dictionary. It contains syntax and guidelines for the HP-IB commands available in the Test Set.

## Notice

Permission to use, copy, and distribute this template is hereby granted, provided that the above copyright notice appears in all copies and that both that copyright notice and this permission notice appear in supporting hardcopy and online documentation. All other rights reserved.

The name of Hewlett-Packard Company or the Hewlett-Packard logo may not be used in advertising or publicity pertaining to distribution of this template without specific, written prior permission.

Hewlett-Packard Company makes no representations about the suitability of this template for any purpose. It is provided “as is” without expressed or implied warranty.

Hewlett-Packard disclaims all warranties with regard to this template, including all implied warranties of merchantability and fitness. In no event shall Hewlett-Packard Company be liable for any special, indirect or consequential damages or any damages whatsoever resulting from loss of use, data or profits, whether in an action of contract, negligence or other tortious action, arising out of or in connection with the use or performance of this template.

## Using This Dictionary

This dictionary is arranged according to HP-IB subsystem. Each subsystem contains syntax for a specific set of functions within the Test Set. With the online version of this document, you can cut and paste the syntax for use in your programs.

The typical format of each entry is as follows:

### **command\_name <parameter>**

Description of command

#### **Syntax**

Use the syntax as a cut-and-paste sample when using the electronic form of this guide. It is set up especially for this purpose, and where possible, includes the limits or expected values for the command.

root:branch1:branch2:branch3:command parameter !limits

#### **Sample Syntax**

root:branch1:branch2:branch3:command 'argument'

## **:AFANalyzer subsystem**

**:AIN <string>**

**:AIN?**

:AIN sets/queries the input state of the Audio In Lo connector.

### **Syntax**

:AFAN:AIN 'Gnd'

'Float'

'600 to Hi'

:AFAN:AIN?

### **Screen/field equivalent**

:AIN sets the Audio In Lo field on the AF Analyzer screen.

**:DEMPHasis <string>**

**:DEMPHasis?**

:DEMP sets/queries the state of the de-emphasis networks in the Audio Analyzer and speaker circuitry.

### **Syntax**

:AFAN:DEMP '750 us'

'Off'

:AFAN:DEMP?

### **Screen/field equivalent**

:DEMP sets the De-Emphasis field on the AF Analyzer screen.

**:DEMPHasis:GAIN <string>**  
**:DEMPHasis:GAIN?**

:DEMP:GAIN sets/queries the AF Analyzer amplifier gain.

**Syntax**

:DEMP:GAIN '0 dB'

'10 dB'

'20 dB'

'30 dB'

:DEMP:GAIN?

**Screen/field equivalent**

:DEMP:GAIN sets the DE-Emp Gain field on the AF Analyzer screen.

**:DETECTOR <string>**  
**:DETECTOR?**

:DET sets/queries the type of detector used for AF signals

**Syntax**

:AFAN:DET 'RMS'

'RMS\*SQRT2'

'PK+'

'PK-'

'PK+/-2'

'PK+-MAX'

'PK+ HOLD'

'PK- HOLD'

'PK+/-2 Hd'

'PK+-MX Hd'

:AFAN:DET?

**Screen/field equivalent**

:DET controls the Detector field in the AF Analyzer screen.

**:DETECTOR:PKLOCATION <string>**  
**:DETECTOR:PKLOCATION?**

:DET:PKL sets/queries the signal source for the peak detector measurements.

**Syntax**

:AFAN:DET:PKL 'Filters'  
                  'De-Emp'  
:AFAN:DET:PKL?

**Screen/field equivalent**

:DET:PKL controls the PK Det To field in the AF ANALYZER screen.

**:DETECTOR:SETTLING <string>**  
**:DETECTOR:SETTLING?**

:DET:SETT sets/queries the settling time for audio measurements. (Use "Fast" when measuring signals greater than 200 Hz).

**Syntax**

:AFAN:DET:SETT 'Fast'  
                  'Slow'  
:AFAN:DET:SETT?

**Screen/field equivalent**

Ext Load R field in the AF ANALYZER screen.

**:ELRESISTOR <real number>**  
**:ELRESISTOR?**

This command sets the external load resistance for measurements using the Audio In connectors.

This command follows the ["Real Number Setting Syntax" on page 210](#), but does not contain the :STATE command.

**Syntax**

:AFAN:ELR <real number> !valid from 1 ohm to 1 megohm  
:AFAN:ELR !returns the value in ohms

**Screen/field equivalent**

Ext Load R field is found on the AF Analyzer screen.

**:FILT1 | FILTER1 <string>**  
**:FILT1? | FILTER1?**

These commands set/query the audio filter 1 selection.

**Syntax**

:AFAN:FILT1 '<20Hz HPF'  
'50Hz HPF'  
'300Hz HPF'  
'Optional Filter specifiers if installed'  
:AFAN:FILT1?

**Screen/field equivalent**

Filter 1 field is found on the AF Analyzerscreen

**:FILT2 | FILTER2 <string>**  
**:FILT2? | FILTER2?**

These commands set/query the audio filter 2 selection.

**Syntax**

:AFAN:FILT2 '300Hz LPF'  
'3kHz LPF'  
'15kHz LPF'  
'>99kHz LPF'  
'Optional Filter specifiers if installed'  
:AFAN:FILT2?

**Screen/field equivalent**

Filter 2 field is found on the AF Analyzer screen.

**:GTIMe <real number>**  
**:GTIMe?**

This command sets the gate time (how long the AF counter samples the input signal) for the audio frequency counter.

This command follows the “[Real Number Setting Syntax](#)” on page 210, but does not contain the :STATe command.

**Syntax**

:AFAN:GTIM <real number> !valid from 10 mS to 1 S

:AFAN:GTIM?

**Screen/field equivalent**

The AF Cnt Gate field is found on the AF Analyzer screen.

**:INPut <string>**  
**:INPut?**

These commands set/query the input to the audio analyzer.

**Syntax**

:AFAN:INP 'FM Demod'

'AM Demod'

'SSB Demod'

'Audio In'

'Ext Mod'

'FM Mod'

'AM Mod'

'Audio Out'

:AFAN:INP?

**Screen/field equivalent**

The AF Anl In field is found on the AF Analyzerscreen.



**:INPut:GAIN <string>**  
**:INPut:GAIN?**

These commands set/query the input gain setting for the audio analyzer.

**Syntax**

:AFAN:INP:GAIN '0 dB'

'20 dB'

'40 dB'

:AFAN:INP:GAIN

**Screen/field equivalent**

Input Gain field is found on the AF Analyzerscreen.

**:SMPoint <string>**  
**:SMPoint?**

These commands set/query the signal source for the oscilloscope.

**Syntax**

:AFAN:SMP 'De-Emp'

'Filters'

'Input'

'Notch'

:AFAN:SMP?

**Screen/field equivalent**

The Scope To field is found on the AF Analyzerscreen.

**:NOTCh:GAIN <string>**  
**:NOTCh:GAIN?**

These commands set/query the gain of the AF Analyzer's notch filter amplifier (used for making SINAD measurements).

**Syntax**

:AFAN:NOTC:GAIN '0 dB'

'10 dB'

'20 dB'

'30 dB'

'40 dB'

:AFAN:NOTC:GAIN?

**Screen/field equivalent**

The Notch Gain field is found on the AF Analyzerscreen.

**:NOTCh:FREQuency <real number>**  
**:NOTCh:FREQuency?**

This command sets the center frequency for the variable frequency notch filter.

.This command follows the ["Real Number Setting Syntax"](#) on page 210, but does not contain the :STATe command.

**Syntax**

:AFAN:NOTC:FREQ <real number> !valid 330 Hz to 10 kHz

:AFAN:NOTC:FREQ?

**Screen/field equivalent**

The Notch Freq field is found on the AF Analyzer screen.

**:RANGing <string>**  
**:RANGing?**

These commands set/query the gain control mode of the AF Analyzer's notch filter amplifier (used for making SINAD measurements).

**Syntax**

:AFAN:RANG 'Auto'

'Hold'

:AFAN:RANG?

**Screen/field equivalent**

The Notch Gain field is found on the AF Analyzerscreen.

**:SPEaker:MODE <string>**  
**:SPEaker:MODE?**

These commands set/query the gain control mode of the AF Analyzer's notch filter amplifier (used for making SINAD measurements).

**Syntax**

:AFAN:SPE:MODE 'On'

'Off'

:AFAN:SPE:MODE?

**Screen/field equivalent**

The Notch Gain field is found on the AF Analyzerscreen.

**:SPEaker:VOLume <string>**  
**:SPEaker:VOLume?**

These commands set/query the speaker volume, which is controlled by the

VOLUME knob when “Pot” is selected.

**Syntax**

:AFAN:SPE:VOL 'Pot'

    'Off'

:AFAN:SPE:VOL

**Screen/field equivalent**

TheSpeaker Vol field is found on the AF Analyzer screen.

## **:AFG1 | AFGENERATOR1 subsystem**

**:AM <real number>**

**:AM?**

This command follows the “[Real Number Setting Syntax](#)” on page 210.

### **Syntax**

:AFG1:AM <real number>

:AFG1:AM? !returns the value of the amplitude modulation

**Screen/field equivalent**

**:DESTination <string>**

**:DESTination?**

:DEST sets/queries the destination of the AFG1 signal.

### **Syntax**

:AFG1:DEST 'AM'

'FM'

'Audio Out'

:AFG1:DEST?

**Screen/field equivalent**

**:FM <real number>**

**:FM?**

This command follows the “[Real Number Setting Syntax](#)” on page 210.

### **Syntax**

:AFG1:FM <real number>

:AFG1:FM?

**Screen/field equivalent**

**:FREQuency <real number>**  
**:FREQuency?**

This command follows the “[Real Number Setting Syntax](#)” on page 210, not including the :STAT command.

**Syntax**

:AFG1:FREQ <real number>

:AFG1:FREQ

**Screen/field equivalent**

**:OUTPut <real number>**  
**:OUTPut?**

This command follows the “[Real Number Setting Syntax](#)” on page 210

**Syntax**

:AFG1:OUTP <real number>

:AFG1:OUTP?

**Screen/field equivalent**

---

## :AFG2 | AFGENERATOR2 subsystem

**:AM <real number>**

**:AM?**

This command follows the “[Real Number Setting Syntax](#)” on page 210.

### **Syntax**

:AFG2:AM <real number>

:AFG2:AM? !returns the value of the amplitude modulation

### **Screen/field equivalent**

**:DESTination <string>**

**:DESTination?**

:DEST sets/queries the destination of the AFG1 signal.

### **Syntax**

:AFG2:DEST 'AM'

'FM'

'Audio Out'

:AFG2:DEST?

### **Screen/field equivalent**

**:FM <real number>**

**:FM?**

This command follows the “[Real Number Setting Syntax](#)” on page 210.

### **Syntax**

:AFG2:FM <real number>

:AFG2:FM?

### **Screen/field equivalent**

**:FREQuency <real number>**  
**:FREQuency?**

This command follows the “[Real Number Setting Syntax](#)” on page 210, not including the :STAT command.

**Syntax**

:AFG2:FREQ <real number>

:AFG2:FREQ

**Screen/field equivalent**

**:OUTPut <real number>**  
**:OUTPut?**

This command follows the “[Real Number Setting Syntax](#)” on page 210

**Syntax**

:AFG2:OUTP <real number>

:AFG2:OUTP?

**Screen/field equivalent**



---

## :CANalyzer subsystem

The :CANalyzer subsystem controls functions unique to the CDMA Analyzer screen.

### :ARM

:ARM arms the measurements based on the type of measurement (single or continuous), qualifying events, and triggering events.

To put the CDMA Analyzer in Single triggering mode, use the :TRIG:MODE:RETR Single command. For Continuous triggering, use :TRIG:MODE:RETR Continuous.

#### Syntax

:CANalyzer:ARM !arms the CDMA analyzer measurements

#### Example

:TRIG:MODE:RETR Single !sets the trigger mode to Single

:CAN:ARM !arms the measurement

#### Screen/field equivalent

:ARM controls the Arm Meas portion of the Analyzer field.

### :AUTO:GAIN?

:AUTO:GAIN? returns the gain setting. The value and mode of the gain are set by the commands :PATH:GAIN and :PATH:GAIN:MODE. This query is valid only while measuring Rho, EVM or Channel Power.

#### Syntax

:CAN:AUTO:GAIN?

#### Screen/field equivalent

:AUTO:GAIN? queries the Gain field of the CDMA Analyzer screen.

## **:AUTO:POWer:GAIN?**

:AUTO:POWer:GAIN? returns the gain setting. The value and mode of the gain are set by the commands :POW:GAIN and :POW:GAIN:MODE. This query is valid only while measuring Average Power.

### **Syntax**

:CAN:AUTO:POW:GAIN?

### **Screen/field equivalent**

This queries the Pwr Gain field of the CDMA Analyzer screen while measuring Average Power.

## **:CHANnel:POWer:CALibrate**

:CHANnel:POWer:CALibrate calibrates the channel power measurement. The Test Set may require a few moments to complete this, so polling of the status register will allow you to determine when the measurement is finished.

### **Syntax**

:CAN:CHAN:POW:CAL

### **Example**

```
:CAN:CHAN:POW:CAL !calibrates channel power
```

### **Screen/field equivalent**

:CHAN:POW:CAL controls the Chn Pwr Cal field of the CDMA Analyzer screen while measuring Channel Power.

## **:DARM**

:DARM works with the :ARM command to control the measurements.

### **Syntax**

:CAN:DARM

### **Screen/field equivalent**

:DARM controls the Disarm portion of the Analyzer field.

**:DIRection <string>**  
**:DIRection?**

:DIR controls the direction of the CDMA analyzer. This command is only valid when the Test Set is in the Frequency mode (see [":CONFigure subsystem" on page 137](#) for the :RFDisplay command to set the mode). :DIR is not used while measuring Average Power.

**Syntax**

```
:CAN:DIR 'Fwd'
      'Rev'
:CAN:DIR?
```

**Screen/field equivalent**

:DIR refers to the Anl Dir field of the CDMA Analyzer screen.

**:EVENT:QUALifier <string>**  
**:EVENT:QUALifier?**

:EVENT:QUAL sets the qualifying event for the trigger. The qualifying event is the event that must occur before a trigger is accepted. The trigger event is set by the :EVENT:TRIG command.

**Syntax**

```
:CAN:EVENT:QUAL 'None'
                '27 ms'
                '20 ms'
                '80 ms'
                '2 s'
                'Ampl Lo'
                'Ampl Mid'
                'Ampl Hi'
                'External'
:CAN:EVENT:QUAL?
```

**Screen/field equivalent**

:EVENT:QUAL sets the Qual Event field in the CDMA Analyzer screen.

## **:EVENT:TRIGger <string>** **:EVENT:TRIGger?**

:EVEN:TRIG sets/queries the Trigger setting of the CDMA Analyzer screen.

### **Syntax**

:CAN:EVEN:TRIG '27 ms'

'20 ms'

'80 ms'

'2 s'

'Delay'

'Immed'

:CAN:EVEN:TRIG?

### **Example**

:CAN:EVEN:TRIG 'Delay' !sets the trigger to run after the delay has been set by the :CAN:EVEN:TRIG:DEL command.

:CAN:EVEN:TRIG '2 s' !sets the trigger to run after 2 seconds

### **Screen/field equivalent**

:EVEN:TRIG sets the Trig Event field of the CDMA Analyzer screen.

## **:EVENT:TRIGger:DELAy <real number>** **:EVENT:TRIGger:DELAy?**

:EVEN:TRIG:DEL sets the delay value of the trigger. This command follows the [“Real Number Setting Syntax” on page 210](#), but does not contain the :DUNits, :UNITs, or :STATe commands.

### **Syntax**

:CAN:EVEN:TRIG:DEL <real number> !values from 20 us to 10 s

:CAN:EVEN:TRIG:DEL? !returns the value set

### **Screen/field equivalent**

**:MODE <string>**  
**:MODE?**

:MODE sets/queries the measurement displayed in the upper left corner of the display.

**Syntax**

:CAN:MODE 'Avg Pwr'

'Rho'

'EVM'

'Chan Pwr'

:CAN:MODE?

**Screen/field equivalent**

:MODE sets the measurement in the upper left corner of the display.

**:PATH:GAIN <string>**  
**:PATH:GAIN?**

:PATH:GAIN sets/queries the gain of the CDMA Analyzer. This command is replaced by the :CAN:POW:GAIN command when Average Power is selected.

**Syntax**

:CAN:PATH:GAIN '0 dB'

'6 dB'

'12 dB'

'18 dB'

'24 dB'

'30 dB'

'36 dB'

:CAN:PATH:GAIN?

**Screen/field equivalent**

:PATH:GAIN sets the Gain field of the CDMA Analyzer screen.

**:PATH:GAIN:MODE <string>**  
**:PATH:GAIN:MODE?**

:PATH:GAIN:MODE sets/queries the mode of the Gain field, allowing you to choose between automatic gain or fixed gain (Hold). This command is replaced by the :CAN:POW:GAIN:MODE command when Average Power is selected.

**Syntax**

:CAN:PATH:GAIN:MODE 'Auto'

'Hold'

:CAN:PATH:GAIN:MODE?

**Screen/field equivalent**

:PATH:GAIN:MODE sets the Auto/Hold portion of the Gain field of the CDMA Analyzer screen.

**:PNOffset <real number>**  
**:PNOffset?**

:PNOffset sets the PN Offset of the CDMA channel being analyzed. This command follows the [“Real Number Setting Syntax” on page 210](#), but does not contain the :STATe command.

**Syntax**

:CAN:PNOF <real number> !values from 0.0 to 511.984375

:CAN:PNOF? !returns the value of the PN Offset

**Screen/field equivalent**

This command controls the PN Offset field in the CDMA Analyzer screen.

**:POWer:GAIN <string>**  
**:POWer:GAIN?**

:POW:GAIN sets/queries the gain of the CDMA Analyzer. This command valid only when Average Power is selected. Otherwise, it is replaced by the :CAN:PATH:GAIN command.

**Syntax**

:CAN:POW:GAIN '0 dB'

'6 dB'

'12 dB'

'24 dB'

'30 dB'

'36 dB'

'42 dB'

'48 dB'

'54 dB'

'60 dB'

'66 dB'

'72 dB'

:CAN:POW:GAIN?

**Screen/field equivalent**

:POW:GAIN sets the Pwr Gain field of the CDMA Analyzer screen, when Avg Pwr is displayed.

## **:POWer:GAIN:MODE <string>** **:POWer:GAIN:MODE?**

:POW:GAIN:MODE sets/queries the mode of the Pwr Gain field, allowing you to choose between automatic gain or fixed gain (Hold). This command is valid only when Average Power is selected. Otherwise, it is replaced by the :CAN:PATH:GAIN:MODE command.

### **Syntax**

:CAN:POW:GAIN:MODE 'Auto'  
'Hold'

:CAN:POW:GAIN:MODE?

### **Screen/field equivalent**

:POW:GAIN:MODE sets the Auto/Hold portion of the Pwr Gain field of the CDMA Analyzer screen, when Avg Pwr is displayed.

## **:POWer:SAMPle:TIME <real number>** **:POWer:SAMPle:TIME?**

:POW:SAMP:TIME sets/queries the Pwr Intvl the amount of time that Average Power or Channel Power is measured. This command follows the [“Real Number Setting Syntax” on page 210](#), but does not contain the :STATE command.

### **Syntax**

:CAN:POW:SAMP:TIME <real number> !values from 0.00025 to .02666 s

:CAN:POW:SAMP:TIME?

### **Screen/field equivalent**

:POW:SAMP:TIME sets the Pwr Intvl field of the CDMA Analyzer screen.

## **:POWer:ZERO**

:POW:ZERO zeroes the power meter.

### **Syntax**

:CAN:POW:ZERO

### **Screen/field equivalent**

:POW:ZERO performs the same command as selecting the Pwr Zero field in the CDMA Analyzer screen.



**:POWer:ZERO:MODE <string>**  
**:POWer:ZERO:MODE?**

:POW:ZERO:MODE controls whether the power meter is automatically zeroed or if the zero function must be manually activated.

**Syntax**

:CAN:POW:ZERO:MODE 'Auto'  
'Manual'

:CAN:POW:ZERO:MODE?

**Screen/field equivalent**

:POW:ZERO:MODE controls the Auto Zero field in the CDMA Analyzer screen.

**:SAMPlE:TIME <real number>**  
**:SAMPlE:TIME?**

:SAMP:TIME sets/queries the amount of time that Rho or EVM is measured. This command follows the [“Real Number Setting Syntax” on page 210](#), but does not contain the :STATe command.

**Syntax**

:CAN:SAMP:TIME <real number> !values from 0.00025 to 0.02 sec for Forward Analyzer Direction ! values from 0.0025 to 0.01 sec for Reverse Channel Direction

:CAN:SAMP:TIME?

**Screen/field equivalent**

:CAN:SAMP:TIME sets the Meas Intvl field of the CDMA Analyzer screen.

**:SPECial <string>**  
**:SPECial?**

:SPEC controls the phase of the I and Q signals are analyzed by the Test Set.

**Syntax**

:CAN:SPEC 'Normal' !set for IS-95 standard rotation  
          'Inverted'

:CAN:SPEC?

**Screen/field equivalent**

:SPEC controls the setting of the Anl Special field of the CDMA Analyzer.

**:TRIGger:STATe <string>**  
**:TRIGger:STATe?**

:TRIG:STAT controls the state of the trigger

**Syntax**

:CAN:TRIG:STAT 'Single'  
                  'Cont'

:CAN:TRIG:STAT?

**Screen/field equivalent**

:TRIG:STAT is used with the Analyzer field of the CDMA Analyzer screen.

---

## :CBuffer subsystem

:CBUF controls the CDMA Data Buffer. This system allows you to send CDMA data through the CDMA Generator. You can send all zeroes, external data via the DATA IN connector, Random data, or data supplied via HP-IB.

### :DATA <48 or 72 character hex string> :DATA?

:DATA is the actual data string sent in the CDMA data buffer. The data buffer is selected by the Data Source field. The command :CGEN:DATA:SOURce in the “:CGENERator subsystem” on page 134 selects the data buffer as the data source.

#### Syntax

:CBUF:DATA <hex string> !48 or 72 character hex string  
:CBUF:DATA?

#### Screen/field equivalent

:DATA is controlled by the Data Source field of the CDMA Generator screen, but :DATA has no direct screen equivalent.

### :FRAME:COUNT <integer>,<real number> :FRAME:COUNT?

:FRAM:COUN sets/queries the number of frames to be transmitted. It is applicable no matter what the source of data is (data buffer, zeroes, external, or random). This command uses the “Multiple Real Number Setting Syntax” on page 192, but without the :DUNits and :UNITs commands.

#### Syntax

:CBUF:FRAM:COUN <integer>,<real number>  
:CBUF:FRAM:COUN <integer> !returns the count corresponding to ‘integer’

#### Screen/field equivalent

:FRAM:COUN is used with the # of Frames field of the CDMA Generator screen.

**:FRAMe:STARt <integer>,<real number>**  
**:FRAMe:STARt?**

:FRAM:STAR sets/queries the first frame of data to be sent. This command uses the [“Multiple Real Number Setting Syntax” on page 192](#), but without the :DUNits and :UNITs commands.

**Syntax**

:FRAM:STAR <integer>,<real number>

:FRAM:STAR?

**Screen/field equivalent**

:FRAM:STAR is used with the Start Frame field of the CDMA Generator screen.

**:FRAMe:LOAD <integer>,<real number>**  
**:FRAMe:LOAD?**

:FRAM:LOAD loads the data into the HP-IB buffer. This command uses the [“Multiple Real Number Setting Syntax” on page 192](#), but without the :DUNits and :UNITs commands.

**Syntax**

:CBuffer:FRAM:LOAD <integer>,<real number>

:CBuffer:FRAM:LOAD?

**Screen/field equivalent**

:FRAM:LOAD has no equivalent field.

**:MODE <string>**  
**:MODE?**

:MODE sets/queries the mode of the Data buffer.

**Syntax**

:CBuffer:MODE 'Single'

'Cont'

:CBuffer:MODE?

**Screen/field equivalent**

:MODE is used with the Data Source field of the CDMA Generator screen.

**:STATe <string>**  
**:STATe?**

:STAT controls the mode of the data buffer.

**Syntax**

:CBUF:STAT 'Idle'

'Send'

:CBUF:STAT?

**Screen/field equivalent**

:STAT is used with the Data Source field of the CDMA Generator screen.

## **:CCOMmon subsystem**

**:PATH <string>**

**:PATH?**

:PATH sets/queries RF Path of the CDMA Generator.

### **Syntax**

:CCOM:PATH 'Bypass'

'IQ'

:CCOM:PATH?

### **Screen/field equivalent**

:PATH is used with the CW RF Path field of the CDMA Generator screen.

---

## **:CDANalyzer subsystem**

:CDAN controls the Code Domain Analyzer.

### **:CONTrol <string>**

### **:CONTrol?**

:CONT sets/queries the menu display of the Code Domain Analyzer.

#### **Syntax**

:CDAN:CONT 'Main'

'Marker'

'Trigger'

'Aux'

'Gain'

'Refs'

'FP Setup'

:CDAN:CONT?

#### **Screen/field equivalent**

:CONT is used with the Control field of the Code Domain Analyzer screen.

## **:EVENT:QUALifier <string>** **:EVENT:QUALifier?**

:EVENT:QUAL sets the qualifying event for the trigger. The qualifying event is the event that must occur before a trigger is accepted. The trigger event is set by the :EVENT:TRIG command.

### **Syntax**

:CDAN:EVENT:QUAL 'None'

'27 ms'

'20 ms'

'80 ms'

'2 s'

'Ampl Lo'

'Ampl Mid'

'Ampl Hi'

'External'

### **Screen/field equivalent**

:EVENT:QUAL is used with the Qual Event field of the Code Domain Analyzer screen.

## **:EVENT:TRIGger <string>** **:EVENT:TRIGger?**

:EVENT:TRIG sets/queries the Trigger setting of the Code Domain Analyzer screen.

### **Syntax**

:CDAN:EVENT:TRIG '27 ms'

'20 ms'

'80 ms'

'2 s'

'Delay'

'Immed'

:CDAN:EVENT:TRIG?

### **Screen/field equivalent**

:EVENT:TRIG sets the Trig Event field of the Code Domain Analyzer screen.



**:EVENT:TRIGger:DELay <real number>**  
**:EVENT:TRIGger:DELay?**

:EVENT:TRIG:DEL sets the delay value of the trigger. This command follows the “[Real Number Setting Syntax](#)” on page 210, but does not contain the :DUNits, :UNITs, or :STATe commands.

**Syntax**

:CDAN:EVENT:TRIG:DEL <real number> !values from 20 us to 10 s  
:CDAN:EVENT:TRIG:DEL? !returns the value set

**Screen/field equivalent**

:EVENT:TRIG:DEL controls the Trig Event Delay field of the Code Domain Analyzer.

**:MARKer:POSition <real number>**  
**:MARKer:POSition?**

:MARK:POS sets/queries the position of the Marker.

**Syntax**

:CDAN:MARK:POS <real number> !values from 0 to 63  
:CDAN:MARK:POS?

**Screen/field equivalent**

:MARK:POS is used with the Walsh Chan field of the Code Domain Analyzer screen.

**:MEASure <string>**  
**:MEASure?**

:MEAS sets/queries the measurement performed in the Code Domain Analyzer screen.

**Syntax**

:CDAN:MEAS 'Power'

'Fast Pwr'

'Timing'

'Phase'

:CDAN:MEAS?

**Screen/field equivalent**

:MEAS is used with the Measurement field of the Code Domain Analyzer screen.

**:FPOWER:NAVG <integer>**  
**:FPOWER:NAVG?**

:FPOWER:NAVG sets/queries the number of averages made by the Fast Power measurement.

**Syntax**

:CDAN:FPOWER:NAVG <integer>

**Screen/field equivalent**

:FPOWER:NAVG is used with the Num Avgs field in the FP Setup menu of the Code Domain Analyzer screen.

**:FPOWer:TOFFset <real number>**  
**:FPOWer:TOFFset?**

:FPOW:TOFF sets/queries the Time Offset for the Fast Power measurement. :FPOW:TOFF can be either entered with this command, or it can be transferred via the :FPOW:TOFF:TRAN command.

This number can be entered in microseconds only.

**Syntax**

:CDAN:FPOW:TOFF <real number> !0 to 26667

:CDAN:FPOW:TOFF?

**Screen/field equivalent**

:FPOW:TOFF is used with the Time Offset field in the Fast Power menu of the Code Domain Analyzer screen.

**:FPOWer:TOFFset:TRANsfer**

:FPOW:TOFF:TRAN transfers the measured Time Offset from the Tm Ofs measurement field to the Time Offset field.

**Syntax**

:CDAN:FPOW:TOFF:TRAN

**Screen/field equivalent**

:FPOW:TOFF:TRAN is used with the Ofs Transfer field in the Fast Power menu of the Code Domain Analyzer screen.

**:MODE <string>**  
**:MODE?**

:MODE sets/queries the measurement state of the Code Domain Analyzer.

**Syntax**

:CDAN:MODE 'Single'

'Cont'

:CDAN:MODE?

**Screen/field equivalent**

:MODE is used with an unnamed field below the Measurement field in the Code Domain Analyzer screen.

**:PATH:GAIN <real number>**  
**:PATH:GAIN?**

:PATH:GAIN sets the Gain in the Gain menu of the Code Domain Analyzer screen.

**Syntax**

:CDAN:PATH:GAIN <real number> !values from 0 to 36 dB

:CDAN:PATH:GAIN?

**Screen/field equivalent**

:PATH:GAIN is used with the GAIN field in the Gain menu of the Code Domain Analyzer screen.

**:POWER:REFERENCE <string>**  
**:POWER:REFERENCE?**

:POW:REF sets the power reference for the Code Domain Analyzer.

**Syntax**

:CDAN:POW:REF '0 dB ref'

:CDAN:POW:REF?

**Screen/field equivalent**

No equivalent field in the Code Domain Analyzer screen.

**:SAMPLE:TIME <real number>**  
**:SAMPLE:TIME?**

:SAMP:TIME sets/queries the time interval for the measurements in the Code Domain Analyzer screen.

**Syntax**

:CDAN:SAMP:TIME <real number> !values 0.00025 to 0.00125 s

:CDAN:SAMP:TIME?

**Screen/field equivalent**

:SAMP:TIME is used with the Interval field in the Aux menu of the Code Domain Analyzer screen.

**:SCALe:PHASe <string>**  
**:SCALe:PHASe?**

:SCAL:PHAS sets/queries the phase portion of the scale that is used when measuring Phase in the Code Domain Analyzer screen.

**Syntax**

:CDAN:SCAL:PHAS '1 mRad'

'2 mRad'

'5 mRad'

'10 mRad'

'20 mRad'

'50 mRad'

:CDAN:SCAL:PHAS?

**Screen/field equivalent**

:SCAL:PHAS is used with the Phase/div field in the Phase measurement menu of the Code Domain Analyzer screen.

**:SCALe:POWer <string>**  
**:SCALe:POWer?**

:SCAL:POW sets/queries the power portion of the scale that is used when measuring Power in the Code Domain Analyzer screen.

**Syntax**

:CDAN:SCAL:POW '1 dB/div'

'2 dB/div'

'5 dB/div'

:CDAN:SCAL:POW?

**Screen/field equivalent**

:SCAL:POW is used with the Pwr Scale field in the Power measurement menu of the Code Domain Analyzer screen.

**:SCAL:TIME <string>**  
**:SCAL:TIME?**

:SCAL:TIME sets/queries the Time/div portion of the scale that is used when measuring Timing in the Code Domain Analyzer screen.

**Syntax**

:CDAN:SCAL:TIME '1 ns'

'2 ns'

'5 ns'

'10 ns'

'20 ns'

'50 ns'

:CDAN:SCAL:TIME?

**Screen/field equivalent**

:SCAL:TIME is used with the Time/div field in the Timing measurement menu of the Code Domain Analyzer screen.

**:THRShld <real number>**  
**:THRShld?**

:THRS sets the minimum display threshold for Code Domain Timing and Phase measurements. Walsh channels that have power levels below the threshold are not displayed in the Timing and Phase displays. Querying these measurements for Walsh Channels that are below the threshold level using HP-IB will return a default value of 9e99.

**Syntax**

:CDAN:THRS <real number> !values minus 30 to 0.0 dB

:CDAN:THRS?

**Screen/field equivalent**

:THRS is used with the Thresold field in the Aux menu of the Code Domain Analyzer screen.

**:TRIGger:ARM**  
**:TRIGger:DARM**

:TRIG:ARM and :TRIG:DARM arm and disarm the measurements in the Code Domain Analyzer screen. This command does not affect Continuous measurement. To put the Code Domain Analyzer in Single triggering mode, use the :TRIG:MODE:RETR Single command. For Continuous triggering, use :TRIG:MODE:RETR Continuous.

**Syntax**

:CDAN:TRIG:ARM !arms the measurement

:CDAN:TRIG:DARM !disarms the measurement that has been armed

**Screen/field equivalent**

:TRIG:ARM and :TRIG DARM are used with the Analyzer field in the Main menu of the Code Domain Analyzer screen.

## **:CGENERator subsystem**

### **:CONTrol <string>**

#### **:CONTrol?**

:CONT defines the CDMA Generator's mode. It defines the type for data being sent.

#### **Syntax**

:CGEN:CONT 'Data'

'Eb/No'

'Noise'

:CGEN:CONT?

#### **Screen/field equivalent**

:CONT is used with the Gen Mode field of the CDMA Generator screen.

### **:DATA:SOURce <string>**

#### **:DATA:SOURce?**

:DATA:SOUR controls the location of the data being sent.

#### **Syntax**

:CGEN:DATA:SOUR 'Zeroes' !Data is all zeroes

'Ext' ! Data comes from the Ext data port

'Random' !Generates random data

'Data Buff' !Gets data from the Data Buffer

:CGEN:DATA:SOUR?

#### **Screen/field equivalent**

:CONT is used with the Gen Mode field of the CDMA Generator screen.



**:DATA:RATE <string>**  
**:DATA:RATE?**

:DATA:RATE sets the data transfer rate for data from the :DATA:SOUR command.

**Syntax**

:CGEN:DATA:RATE '14.4 Kbps'  
                  '9.6 Kbps'  
:CGEN:DATA:RATE?

**Screen/field equivalent**

:DATA:RATE is used with the Data Rate field of the CDMA Generator screen.

**:DIRection <string>**  
**DIRection?**

:DIR sets/queries the direction of the CDMA Generator.

**Syntax**

:CGEN:DIR 'Fwd'  
          'Rev'  
:CGEN:DIR?

**Screen/field equivalent**

:DIR is used with the Gen Dir field of the CDMA Generator screen.

**:EBNO:LEVel <real number>**  
**:EBNO:LEVel?**

:EBNO:LEV sets/queries the level of the generator when it is in Eb/No mode. This command uses the ["Real Number Setting Syntax"](#), without the :STAT command.

**Syntax**

:CGEN:EBNO:LEV <real number> !values from -5 to +25 dB  
:CGEN:EBNO:LEV?

**Screen/field equivalent**

:EBNO:LEV is used with the Gen Mode field of the CDMA Generator screen.

**:EQFilter <string>**  
**:EQFilter?**

:EQF sets/queries the setting of the Equalization Filter. This command is used when you want to override the automatic setting (filter is applied for forward link signals, and bypassed for reverse link signals).

**Syntax**

:CGEN:EQF 'EQ In' !switches the Equalization filter in  
          'Out' !bypasses the Equalization filter

:CGEN:EQF?

**Screen/field equivalent**

:EQF is used with the Gen Dir field of the CDMA Generator screen.

**:EVENsec <string>**  
**:EVENsec?**

:EVEN sets/queries the Even Sec In setting. When enabled, signals applied to the Even Sec In port synchronize the Test Set's timing circuits.

**Syntax**

:CGEN:EVEN 'Enable'  
          'Not'

:CGEN:EVEN?

**Screen/field equivalent**

:EVEN is used with the Even Sec In field of the CDMA Generator screen.

**:SPECial <string>**  
**:SPECial?**

:SPEC sets/queries the Special field of the Generator.

**Syntax**

:CGEN:SPEC 'Normal'  
          'Invert'

:CGEN:SPEC?

**Screen/field equivalent**

:SPEC is used with the Special field of the CDMA Generator screen.

---

## :CONFigure subsystem

The :CONFigure subsystem contains commands that control several different screens: I/O CONFIGURE, INSTRUMENT CONFIGURE, and PRINT CONFIGURE.

### :SRLocation <string>

#### :SRLocation?

:SRL sets/queries the Save/Recall location.

#### Syntax

:CONF:SRL 'INTERNAL'

'CARD'

'RAM'

:CONF:SRL?

#### Screen/field equivalent

:SRL is used with the Save/Recall field of the I/O Configure screen.

### :BADDress <integer>

:BADD sets the HP-IB address for the Test Set. This command follows the ["Integer Number Setting Syntax"](#).

#### Syntax

:CONF:BADD <integer> !valid from 0 to 30

:CONF:BADD?

#### Screen/field equivalent

:BADD is used with the HP-IB Adrs field of the I/O Configure screen.

**:BEEPer <string>**  
**:BEEPer?**

:BEEP sets/queries the setting of the beeper within the Test Set.

**Syntax**

:CONF:BEEP 'Off'

'Quiet'

'Loud'

:CONF:BEEP?

**Screen/field equivalent**

:BEEP is used with the BEEPER field of the Instrument Configure screen.

**:DISPlay:MESSages**  
**:DISPlay:MESSages?**

:DISP:MESS sets/queries the reporting of messages during normal operation. A Yes setting means that all messages are displayed as they occur.

**Syntax**

:CONF:DISP:MESS 'Yes'

'No'

:CONF:DISP:MESS

**Screen/field equivalent**

:DISP:MESS is used with the Display User Messages field of the Instrument Configure screen.

**:BMODe <string>**  
**:BMODe?**

:BMOD sets/queries the HP-IB control mode.

**Syntax**

:CONF:BMOD 'Control'

'Talk&Lstn'

:CONF:BMOD?

**Screen/field equivalent**

:BMOD is used with the Mode field of the I/O Configure screen.

**:DATE <integer>**  
**:DATE?**

:DATE sets/queries the date set within the Test Set. This command uses the “Integer Number Setting Syntax”.

**Syntax**

:CONF:DATE <integer> !Use MMDDYY format

:CONF:DATE?

**Screen/field equivalent**

:DATE is used with the Date field of the Instrument Configure screen.

**:FCLK <string>**  
**:FCLK?**

:FCLK sets/queries the Frame clock output.

**Syntax**

:CONF:FCLK '20.00 ms'

'26.67 ms'

'80.00 ms'

'2.00 s'

:CONF:FCLK?

**Screen/field equivalent**

:FCLK is used with the Frame Clock Output field of the Instrument Configure screen.

**:RFDisplay <string>**  
**:RFDisplay?**

:RFD sets/queries the RF display mode.

**Syntax**

:CONF:RFD 'Freq' !sets RF display to Frequency mode

'Chan' !sets RF display to Channel mode

:CONF:RFD?

**Screen/field equivalent**

:RFD is used with the RF Display field of the Instrument Configure screen.

**:KNOB <string>**  
**:KNOB?**

:KNOB is an HP-IB command which is the equivalent of the pressing the knob.

**Syntax**

:CONF:KNOB 'On'  
          'Off'  
:CONF:KNOB?

**Screen/field equivalent**

:KNOB has no field associated with this command..

**:NOTChmode <string>**  
**:NOTChmode?**

:NOTC sets/queries the coupling between the AF Generator 1 and a notch filter. The frequency and gain of the notch filter used is set by the :AFAN:NOTC:GAIN and :AFAN:NOTC:FREQ fields in the AF Analyzer screen.

**Syntax**

:CONF:NOTC 'AFGen1' !notch filter switched in  
          'None' !no notch filter  
:CONF:NOTC?

**Screen/field equivalent**

:NOTC is used with the Notch Coupl field of the Instrument Configure screen.

**:OFLevel:MODE <string>**  
**:OFLevel:MODE?**

:OFL:MODE sets/queries the RF Level Offset, and is typically used to compensate for path loss between the Test Set and the Base Station.

**Syntax**

:CONF:OFL:MODE 'On'  
                  'Off'

:CONF:OFL:MODE?

**Screen/field equivalent**

:OFL:MODE is used with the RF Level Offset field of the Instrument Configure screen.

**:OFLevel:ANTenna <real number>**  
**:OFLevel:ANTenna?**

:OFL:ANT sets/queries the correction for the path loss from the Device Under Test to the Test Set's Antenna In port. This correction is applied when the RF Level Offset field (:CONF:OFL:MODE) is "On". This command uses the ["Real Number Setting Syntax"](#), without the :STAT command.

**Syntax**

:CONF:OFL:ANT <real number> !values: minus 100.0 to 100.0  
:CONF:OFL:ANT?

**Screen/field equivalent**

:OFL:ANT is used with the RF Level Offset field of the Instrument Configure screen.

## **:OFLevel:DUPLex <real number>**

:OFL:DUP sets/queries the path loss from the Duplex Out connector to the Device Under Test. This correction is applied when the RF Level Offset field is set to "On". This command uses the "Real Number Setting Syntax", without the :STAT command.

### **Syntax**

:CONF:OFL:DUP <real number> !values: minus 100.0 to 100.0

:CONF:OFL:DUP?

### **Screen/field equivalent**

:OFL:DUP is used with the Duplex Out field of the Instrument Configure screen.



## **:OFLevel:RFINout <real number>**

:OFL:RFIN sets/queries the path loss from the RF In/Out connector to the Device Under Test. This correction is applied when the RF Level Offset field is set to “On”. This command uses the “[Real Number Setting Syntax](#)”, without the :STAT command.

### **Syntax**

:CONF:OFL:RFIN <real number> !values: minus 100.0 to 100.0

:CONF:OFL:RFIN?

### **Screen/field equivalent**

:OFL:RFIN is used with the RF In/Out field of the Instrument Configure screen.

## **:OFRequency <real number>**

:OFR sets/queries the Test Set’s frequency offset between the Generator and Analyzer. The RF Display field must be set to “Freq”, and the RF Offset field must be set to “On”. This command uses the “[Real Number Setting Syntax](#)”, without the :STAT command.

### **Syntax**

:CONF:OFR <real number>

:CONF:OFR?

### **Screen/field equivalent**

:OFR is used with the (Gen) - (Anl) field of the Instrument Configure screen.

## **:OMODe <string>**

### **:OMODe?**

:OMOD sets/queries the RF Offset field, which enables or disables the frequency offset defined in the :CONF:OFR command.

### **Syntax**

:CONF:OMOD ‘On’

‘Off’

:CONF:OMOD?

### **Screen/field equivalent**

:OMOD is used with the RF Offset field of the Instrument Configure screen.

## **:OPERation:AUTO** **:OPERation:HOLD**

:OPER:AUTO and :OPER:HOLD control the autoranging and autotuning routines in the Test Set. :AUTO enables the routines, :HOLD disables them.

### **Syntax**

:CONF:OPER:AUTO !enables autotuning and autoranging

:CONF:OPER:HOLD !disables autotuning and autoranging

### **Screen/field equivalent**

:OPER:AUTO and :OPER:HOLD are used with the Range Hold field of the Configure screen.

## **:PRINt:ADdRes <integer>** **:PRINt:ADdRes?**

:PRIN:ADD sets/queries the printer address used when “HP-IB” is selected in the Printer Port field. This command uses the “[Integer Number Setting Syntax](#)”.

### **Syntax**

:CONF:PRIN:ADD <integer> !values 0 to 30

:CONF:PRIN:ADD?

### **Screen/field equivalent**

:PRIN:ADD is used with the Printer Adrs field of the Print Configure screen.

## **:PRINt:LINEs | LINE <integer>** **:PRINt:LINEs | LINE?**

:PRIN:LIN sets/queries the number of lines to be printed per page. This command uses the “[Integer Number Setting Syntax](#)”.

### **Syntax**

:CONF:PRIN:LIN <integer> !values: 5 to 120

:CONF:PRIN:LIN?

### **Screen/field equivalent**

:PRIN:LIN is used with the Lines/Page field of the Print Configure screen.

**:PRINT:DESTination | PORTs <string>**  
**:PRINT:DESTination? | PORTs?**

:PRIN:DEST sets/queries the port setting for the Printer Port field. This command controls the ports available from the keypad (Serial 9, HP-IB, and Parallel 15) . The :PRIN:DEST command is equivalent to the :PRIN:PORT command.

**Syntax**

```
:CONF:PRIN:DEST 'Serial 9'  
      'Parallel 15'  
      'HP-IB'  
:CONF:PRIN:DEST?
```

**Screen/field equivalent**

:PRIN:DEST is used with the Printer Port field of the Print Configure screen.

**:PRINT:FFStart <string>**  
**:PRINT:FFStart?**

:PRIN:FFST sets/queries the Form Feed (FF at Start) field, which determines if there will be a form feed at the start of printing.

**Syntax**

```
:CONF:PRIN:FFST 'Yes'  
      'No'  
:CONF:PRIN:FFST?
```

**Screen/field equivalent**

:PRIN:FFST is used with the FF at Start field of the Print Configure screen.

**:PRINt:FFENd <string>**  
**:PRINt:FFENd?**

:PRIN:FFEN sets/queries the Form Feed (FF at End) field, which determines if there will be a form feed at the end of printing.

**Syntax**

:CONF:PRIN:FFEN 'Yes'

'No'

:CONF:PRIN:FFEN?

**Screen/field equivalent**

:PRIN:FFEN is used with the FF at End field of the Print Configure screen.

**:PRINt:TITLe <string>**  
**:PRINt:TITLe?**

:PRIN:TITL defines the title of the heading of the print output.

Available character set:

ABCDEFGHIJKLMNOPQRSTUVWXYZ

abcdefghijklmnopqrstuvwxyz

\_0123456789 !@#\$%^&\*()-+=<>?[]\ |';",./

**Syntax**

:CONF:PRIN:TITL '<string>' !up to 50 characters

:CONF:PRIN:TITL

**Screen/field equivalent**

:PRIN:TITL is used with the Print Title field of the Print Configure screen.

**:PRINt:PRINter | :PRINt:HPModel | :PRINt:HPMO <string>**  
**:PRINt:PRINter? | :PRINt:HPModel? | :PRINt:HPMO?**

:PRIN:PRIN sets the type of printer chosen for printing.

### Syntax

:CONF:PRIN:PRIN 'ThinkJet'

'QuietJet'

'PaintJet'

'DeskJet'

'LaserJet'

'Epson FX-80'

'Epson LX-850'

:CONF:PRIN:PRIN?

### Screen/field equivalent

:PRIN:PRIN is used with the Model field of the Print Configure screen.

**:REFerence:INPut:SElect <string>**  
**:REFerence:INPut:SElect**

:REF:INP:SEL sets/queries the Reference Input selection. Internal and External define where the input should come from, and Auto will switch between External and Internal, based on the signal presence. You must still set the frequency of the Ext Ref Input even though this is set to Auto. It does not auto-select the frequency.

### Syntax

:CONF:REF:INP:SEL 'Auto'

'Internal'

'External'

:CONF:REF:INP:SEL?

### Screen/field equivalent

:REF:INP:SEL is used with the Ref Select field of the Instrument Configure screen.

## **:REFerence:INPut:EXTernal <string>** **:REFerence:INPut:EXTernal?**

:REF:INP:EXT sets/queries the frequency of an External Reference signal. This must be used in conjunction with the :REF:INP:SEL command.

### **Syntax**

:REF:INP:EXT '1 MHz'

'2 MHz'

'5 MHz'

'10 MHz'

'15 MHz'

'1X Chip'

'2X Chip'

'4X Chip'

'8X Chip'

'16X Chip'

:REF:INP:EXT?

### **Screen/field equivalent**

:REF:INP:EXT is used with the Ext Ref In field of the Instrument Configure screen.

## **:RFIMped <string>** **:RFIMPed?**

:RFIM sets/queries the way that RF Generator voltages are expressed (across a 50 ohm load or open circuit). The Amplitude field units must be set to V, mV, uV, or dbuV for this command to have an effect.

### **Syntax**

:CONF:RFIM '50 ohm'

'emf'

:CONF:RFIM?

### **Screen/field equivalent**

:RFIM is used with the RFGen Volts field of the Instrument Configure screen.

**:SPSelect <string>**  
**:SPSelect?**

:SPS sets/queries the selected serial port for configuration purposes. It defines which port's configuration settings are displayed on the I/O CONFIGURE screen.

**Syntax**

:CONF:SPS 'Serial 9'

'Serial 10'

'Serial 11'

:CONF:SPS?

**Screen/field equivalent**

:SPS is used with the Serial Port field of the I/O Configure screen.

**:SPOR9:BAUD | SPORT9:BAUD <string>**  
**:SPOR9:BAUD? | SPORT9:BAUD?**

:SPOR9:BAUD sets/queries the baud rate of the Serial Port 9.

**Syntax**

:CONF:SPOR9:BAUD '150'  
'300'  
'600'  
'1200'  
'2400'  
'4800'  
'9600'  
'19200'

:CONF:SPOR9:BAUD?

**Screen/field equivalent**

:SPOR9:BAUD is used with the Serial Baud field of the I/O CONFIGURE screen.

**:SPOR9:PARity | SPORT9:PARity <string>**  
**:SPOR9:PARity? | SPORT9:PARity?**

:SPOR9:PARity sets/queries the parity of the Serial Port 9.

**Syntax**

:CONF:SPOR9:PAR 'None'  
'Odd'  
'Even'  
'Always 1'  
'Always 0'

:CONF:SPOR9:PAR?

**Screen/field equivalent**

:SPOR9:PAR is used with the Parity field of the I/O CONFIGURE screen.



**:SPOR9:DATA | SPORT9:DATA <string>**  
**:SPOR9:DATA? | SPORT9:DATA?**

:SPOR9:DATA sets/queries the number of data bits per data word sent over the Serial port 9.

**Syntax**

:CONF:SPOR9:DATA '7 bits'  
                  '8 bits'  
:CONF:SPOR9:DATA?

**Screen/field equivalent**

:SPOR9:DATA is used with the Data Length field of the I/O CONFIGURE screen.

**:SPOR9:STOP | SPORT9:STOP <string>**  
**:SPOR9:STOP? | SPORT9:STOP?**

SPOR9:STOP sets/queries the number of stop bits used for serial communication over the Serial Port 9.

**Syntax**

:CONF:SPOR9:STOP '1 bit'  
                  '2 bits'  
:CONF:SPOR9:STOP?

**Screen/field equivalent**

:SPOR9:STOP is used with the Stop Length field of the I/O CONFIGURE screen.

**:SPOR9:FCONtrol | SPORT9:FCONtrol <string>**  
**:SPOR9:FCONtrol? | SPORT9:FCONtrol?**  
**:SPOR9:FLOW | SPORT9:FLOW <string>**  
**:SPOR9:FLOW? | SPORT9:FLOW?**

:SPOR9:FCON sets/queries the flow control to set the pace of serial communications for the Serial Port 9.

**Syntax**

:CONF:SPOR9:FCON 'Xon/Xoff'  
'None'

:CONF:SPOR9:FCON?

**Screen/field equivalent**

:SPOR9:FCON is used with the Flow Control field of the I/O CONFIGURE screen.

**:SPOR9:SIN | SPORT9:SIN <string>**  
**:SPOR9:SIN? | SPORT9:SIN?**  
**:SPOR9:SINPut | SPORT9:SINPut <string>**  
**:SPOR9:SINPut? | SPORT9:SINPut?**

:SPOR9:SIN sets/queries the destination of data characters received by the Test Set via the Serial Port 9.

**Syntax**

:CONF:SPOR9:SIN 'Inst'  
'IBASIC'

:CONF:SPOR9:SIN?

**Screen/field equivalent**

:SPOR9:SIN is used with the Serial\_9 In field of the I/O CONFIGURE screen.

**:SPOR9:IBEcho | SPORT9:IBEcho <string>**  
**:SPOR9:IBEcho? | SPORT9:IBEcho?**

:SPOR9:IBEC sets/queries the screen and error message echoing from IBASIC.

**Syntax**

:CONF:SPOR9:IBEC 'On'  
'Off'

:CONF:SPOR9:IBEC

**Screen/field equivalent**

:SPOR9:IBEC is used with the IBASIC Echo field of the I/O CONFIGURE screen.

**:SPOR9:IECHo | SPORT9:IECHo <string>**  
**:SPOR9:IECHo? | SPORT9:IECHo?**

:SPOR9:IECH sets/queries the character and screen echoing (instrument echo) when using an external ASCII RS-232 terminal or computer to enter or edit the IBASIC program.

**Syntax**

:CONF:SPOR9:IECH 'On'  
'Off'

:CONF:SPOR9:IECH

**Screen/field equivalent**

:SPOR9:IECH is used with the Inst Echo field of the I/O CONFIGURE screen.

**:SPOR9:STATus:LINE?**

:SPOR9:LINE? queries the Line status register.

**Syntax**

:CONF:SPOR9:STAT:LINE?

**Screen/field equivalent**

No screen/field equivalent.

## **:SPOR9:STATus:MODem?**

:SPOR9:MOD? queries the Modem status register.

### **Syntax**

:CONF:SPOR9:STAT:MOD?

### **Screen/field equivalent**

No screen/field equivalent.

## **:SPOR10:BAUD | SPORT10:BAUD <string> :SPOR10:BAUD? | SPORT10:BAUD?**

:SPOR10:BAUD sets/queries the baud rate of the Serial Port 10.

### **Syntax**

:CONF:SPOR10:BAUD '150'  
'300'  
'600'  
'1200'  
'2400'  
'4800'  
'9600'  
'19200'

:CONF:SPOR10:BAUD?

### **Screen/field equivalent**

:SPOR10:BAUD is used with the Serial Baud field of the I/O CONFIGURE screen.

**:SPOR10:PARity | SPORT10:PARity <string>**  
**:SPOR10:PARity? | SPORT10:PARity?**

:SPOR10:PARity sets/queries the parity of the Serial Port 10.

**Syntax**

:CONF:SPOR10:PAR 'None'

'Odd'

'Even'

'Always 1'

'Always 0'

:CONF:SPOR10:PAR?

**Screen/field equivalent**

:SPOR10:PAR is used with the Parity field of the I/O CONFIGURE screen.

**:SPOR10:DATA | SPORT10:DATA <string>**  
**:SPOR10:DATA? | SPORT10:DATA?**

:SPOR10:DATA sets/queries the number of data bits per data word sent over the Serial port 10.

**Syntax**

:CONF:SPOR10:DATA '7 bits'

'8 bits'

:CONF:SPOR10:DATA?

**Screen/field equivalent**

:SPOR10:DATA is used with the Data Length field of the I/O CONFIGURE screen.

**:SPOR10:STOP | SPORT10:STOP <string>**  
**:SPOR10:STOP? | SPORT10:STOP?**

SPOR10:STOP sets/queries the number of stop bits used for serial communication over the Serial Port 10.

**Syntax**

:CONF:SPOR10:STOP '1 bit'  
                  '2 bits'

:CONF:SPOR10:STOP?

**Screen/field equivalent**

:SPOR10:STOP is used with the Stop Length field of the I/O CONFIGURE screen.

**:SPOR10:FCONtrol | SPORT10:FCONtrol <string>**  
**:SPOR10:FCONtrol? | SPORT10:FCONtrol?**  
**:SPOR10:FLOW | SPORT10:FLOW <string>**  
**:SPOR10:FLOW? | SPORT10:FLOW?**

:SPOR10:FCON sets/queries the flow control to set the pace of serial communications for the Serial Port 10.

**Syntax**

:CONF:SPOR10:FCON 'Xon/Xoff'  
                  'None'

:CONF:SPOR10:FCON?

**Screen/field equivalent**

:SPOR10:FCON is used with the Flow Control field of the I/O CONFIGURE screen.

**:SPOR10:SIN | SPORT10:SIN <string>**  
**:SPOR10:SIN? | SPORT10:SIN?**  
**:SPOR10:SINPut | SPORT10:SINPut <string>**  
**:SPOR10:SINPut? | SPORT10:SINPut?**

:SPOR10:SIN sets/queries the destination of data characters received by the Test Set via the Serial Port 10.

**Syntax**

:CONF:SPOR10:SIN 'Inst'

'IBASIC'

:CONF:SPOR10:SIN?

**Screen/field equivalent**

:SPOR10:SIN is used with the Serial\_10 In field of the I/O CONFIGURE screen.

**:SPOR10:IBECho | SPORT10:IBECho <string>**  
**:SPOR10:IBECho? | SPORT10:IBECho?**

:SPOR10:IBEC sets/queries the screen and error message echoing from IBASIC.

**Syntax**

:CONF:SPOR10:IBEC 'On'

'Off'

:CONF:SPOR10:IBEC

**Screen/field equivalent**

:SPOR10:IBEC is used with the IBASIC Echo field of the I/O CONFIGURE screen.

**:SPOR10:IECHo | SPORT10:IECHo <string>**  
**:SPOR10:IECHo? | SPORT10:IECHo?**

:SPOR10:IECH sets/queries the character and screen echoing (instrument echo) when using an external ASCII RS-232 terminal or computer to enter or edit the IBASIC program.

**Syntax**

:CONF:SPOR10:IECH 'On'  
'Off'

:CONF:SPOR10:IECH

**Screen/field equivalent**

:SPOR10:IECH is used with the Inst Echo field of the I/O CONFIGURE screen.

**:SPOR10:STATus:LINE?**

:SPOR10:LINE? queries the Line status register.

**Syntax**

:CONF:SPOR10:STAT:LINE?

**Screen/field equivalent**

No screen/field equivalent.

**:SPOR10:STATus:MODem?**

:SPOR10:MOD? queries the Modem status register.

**Syntax**

:CONF:SPOR10:STAT:MOD?

**Screen/field equivalent**

No screen/field equivalent.



**:SPOR11:BAUD | SPORT11:BAUD <string>**  
**:SPOR11:BAUD? | SPORT11:BAUD?**

:SPOR11:BAUD sets/queries the baud rate of the Serial Port 11.

**Syntax**

:CONF:SPOR11:BAUD '150'  
    '300'  
    '600'  
    '1200'  
    '2400'  
    '4800'  
    '9600'  
    '19200'

:CONF:SPOR11:BAUD?

**Screen/field equivalent**

:SPOR11:BAUD is used with the Serial Baud field of the I/O CONFIGURE screen.

**:SPOR11:PARity | SPORT11:PARity <string>**  
**:SPOR11:PARity? | SPORT11:PARity?**

:SPOR11:PARity sets/queries the parity of the Serial Port 11.

**Syntax**

:CONF:SPOR11:PAR 'None'  
    'Odd'  
    'Even'  
    'Always 1'  
    'Always 0'

:CONF:SPOR11:PAR?

**Screen/field equivalent**

:SPOR11:PAR is used with the Parity field of the I/O CONFIGURE screen.

**:SPOR11:DATA | SPORT11:DATA <string>**  
**:SPOR11:DATA? | SPORT11:DATA?**

:SPOR11:DATA sets/queries the number of data bits per data word sent over the Serial port 11.

**Syntax**

:CONF:SPOR11:DATA '7 bits'  
                  '8 bits'

:CONF:SPOR11:DATA?

**Screen/field equivalent**

:SPOR11:DATA is used with the Data Length field of the I/O CONFIGURE screen.

**:SPOR11:STOP | SPORT11:STOP <string>**  
**:SPOR11:STOP? | SPORT11:STOP?**

SPOR11:STOP sets/queries the number of stop bits used for serial communication over the Serial Port 11.

**Syntax**

:CONF:SPOR11:STOP '1 bit'  
                  '2 bits'

:CONF:SPOR11:STOP?

**Screen/field equivalent**

:SPOR11:STOP is used with the Stop Length field of the I/O CONFIGURE screen.

**:SPOR11:FCONtrol | SPORT11:FCONtrol <string>**  
**:SPOR11:FCONtrol? | SPORT11:FCONtrol?**  
**:SPOR11:FLOW | SPORT11:FLOW <string>**  
**:SPOR11:FLOW? | SPORT11:FLOW?**

:SPOR11:FCON sets/queries the flow control to set the pace of serial communications for the Serial Port 11.

**Syntax**

:CONF:SPOR11:FCON 'Xon/Xoff'

'None'

:CONF:SPOR11:FCON?

**Screen/field equivalent**

:SPOR11:FCON is used with the Flow Control field of the I/O CONFIGURE screen.

**:SPOR11:SIN | SPORT11:SIN <string>**  
**:SPOR11:SIN? | SPORT11:SIN?**  
**:SPOR11:SINPut | SPORT11:SINPut <string>**  
**:SPOR11:SINPut? | SPORT11:SINPut?**

:SPOR11:SIN sets/queries the destination of data characters received by the Test Set via the Serial Port 11.

**Syntax**

:CONF:SPOR11:SIN 'Inst'

'IBASIC'

:CONF:SPOR11:SIN?

**Screen/field equivalent**

:SPOR11:SIN is used with the Serial\_11 In field of the I/O CONFIGURE screen.

**:SPOR11:IBEcho | SPORT11:IBEcho <string>**  
**:SPOR11:IBEcho? | SPORT11:IBEcho?**

:SPOR11:IBEC sets/queries the screen and error message echoing from IBASIC.

**Syntax**

:CONF:SPOR11:IBEC 'On'  
'Off'

:CONF:SPOR11:IBEC

**Screen/field equivalent**

:SPOR11:IBEC is used with the IBASIC Echo field of the I/O CONFIGURE screen.

**:SPOR11:IECHo | SPORT11:IECHo <string>**  
**:SPOR11:IECHo? | SPORT11:IECHo?**

:SPOR11:IECH sets/queries the character and screen echoing (instrument echo) when using an external ASCII RS-232 terminal or computer to enter or edit the IBASIC program.

**Syntax**

:CONF:SPOR11:IECH 'On'  
'Off'

:CONF:SPOR11:IECH

**Screen/field equivalent**

:SPOR11:IECH is used with the Inst Echo field of the I/O CONFIGURE screen.

**:SPOR11:STATus:LINE?**

:SPOR11:LINE? queries the Line status register.

**Syntax**

:CONF:SPOR11:STAT:LINE?

**Screen/field equivalent**

No screen/field equivalent.

## **:SPOR11:STATus:MODem?**

:SPOR11:MOD? queries the Modem status register.

### **Syntax**

:CONF:SPOR11:STAT:MOD?

### **Screen/field equivalent**

No screen/field equivalent.

## **:TBASE:CDMA <string>**

## **:TBASE:CDMA?**

:TBASE:CDMA sets/queries the optional time base used for CDMA. This commands allows you to select either an external or internal timebase.

### **Syntax**

:CONF:TBASE:CDMA 'Internal'

'External'

:CONF:TBASE:CDMA?

### **Screen/field equivalent**

:TBASE:CDMA is used with the Opt CDMA TB field of the INSTRUMENT CONFIGURE screen.

**:TIME <real number>**  
**:TIME?**

:TIME sets/queries the time of day. This command uses the “[Real Number Setting Syntax](#)”, without the :STAT, :DUN, :INCR, or :UNITs command.

**Syntax**

:CONF:TIME <real number> !hh.mm where hh is hours, mm is minutes

:CONF:TIME?

**Screen/field equivalent**

:TIME is used with the Time field of the INSTRUMENT CONFIGURE screen.

---

## **:DISPlay subsystem**

The :DISPlay subsystem controls the display of screens. A screen must be displayed before a measurement from that screen can be made.

### **:DISPlay :DISPlay?**

:DISP sets/queries which screen is displayed.

#### **Syntax**

```
:DISP AFANalyzer !AF Analyzer screen
      CANalyzer !CDMA Analyzer
      CDANalyzer !Code Domain Analyzer
      CONFigure !Inst Configure
      HELP !Help screen
      IOConfigure !I/O Configure
      MESSages !Messages screen
      OSCilloscope
      PCONfigure !Print Configure
      RFANalyzer !RF Analyzer
      RFGen !RF Generator screen
      SANalyzer !Spectrum Analyzer
      SERvice
      TCONfigure !TESTS External devices
      TESTs !TESTS screen
      TEXec !TESTS Execution screen
      TFReq !TESTS Channel Information screen
      THLP !TESTS Help screen
      TIBasic !TESTS IBASIC screen
      TMAKe !TESTS Save/Delete Procedure
      TPARm !TESTS Test Parameters
      TPRint !TESTS print screen
      TSEQn !TESTS Order of Tests
```

**TSPec !TESTS PassFail Limits**

**CGENerator !CDMA Generator**

**REMOte !locks the display**

**:DISP? !returns the name of the screen being displayed**



## Integer Number Setting Syntax

This syntax is for setting values of commands which require integer values. It is to be used with commands which call for the Integer Number Setting Syntax.

An example of a command that requires the Integer Setting Syntax is the :RFANalyzer:RFCHannel command (see "[:RFANalyzer subsystem](#)" on page 215).

### Syntax

*:Previous Syntax* <integer\_value> !decimal value

*:Previous Syntax* #B<Binary\_integer\_value> !maximum 32 bits

*:Previous Syntax* #O<Octal\_integer\_value>

*:Previous Syntax* #H<Hexadecimal\_integer\_value>

*:Previous Syntax*:INCRement UP !Increments the present value

*:Previous Syntax*:INCRement DOWN !Decrements the present value

*:Previous Syntax?* !Query Returns the present value

#### Examples

:RFAN:RFCH 47

:RFAN:RFCH #B101111 !sets channel to 47 binary

:RFAN:RFCH #O57 !sets channel to 47 octal

:RFAN:RFCH #H2F !sets channel to 47 hexadecimal

:RFAN:RFCH:INCR UP !increments by 1

:RFAN:RFCH:INCR DOWN !decrements by 1

## :MEASure subsystem

The MEAS subsystem has a set of commands unique to this subsystem. These are the <measurement command> set. They control features of the Test Set such as setting measurement limits, units and the meters. See “[Number Measurement Syntax](#)” on [page 193](#) for more details about these commands.

Most commands have both a <measurement command> parameter and a query command. The command with the <measurement command> allows you to set certain parameters of the measurement such as limits, averages, and units. The query command is the command that actually reads the value.

## :RESet

Resets all measurements in progress.

### Syntax

:MEAS:RESET

## :AFRequency:ACLevel <measurement command> :AFRequency:ACLevel?

:AFR:ACL controls/reads the AC level measurement. It measures the AC level of the Audio source as selected by the :AFAN:INP command [page 102](#).

### Syntax

:MEAS:AFR:ACL <meas cmd> !see “[Number Measurement Syntax](#)” on [page 193](#)

:MEAS:AFR:ACL? !returns real value

### Screen/field equivalent

:AFR:ACL reads the AC Level measurement field on the AF ANALYZER and RF ANALYZER screens.

**:AFRequency:AM <measurement command>**  
**:AFRequency:AM?**

:AFR:AM controls/reads the AM depth of modulation measurement.

**Syntax**

:MEAS:AFR:AM <meas cmd> !see "[Number Measurement Syntax](#)" on page 193

:MEAS:AFR:AM? !returns real value

**Example**

:MEAS:AFR:AM:METer ON !displays the measurement in the meter

**Screen/field equivalent**

:AFR:AM reads the AM measurement field on the AF ANALYZER screen.

**:AFRequency:CURRent <measurement command>**  
**:AFRequency:CURRent?**

:AFR:CURR controls/reads the Current measurement.

**Syntax**

:MEAS:AFR:CURR <meas cmd> !see "[Number Measurement Syntax](#)" on page 193

:MEAS:AFR:CURR? !returns real value

**Example**

:MEAS:AFR:CURR? !displays the current measurement

**Screen/field equivalent**

:AFR:CURR reads the Current measurement field on the AF ANALYZER and RF ANALYZER screens.

## **:AFRequency:DCAM <measurement command>** **:AFRequency:DCAM?**

:AFR:DCAM controls/reads the DC AM measurement. To use this measurement, you must have the DC Level measurement selected (see :MEAS:AFR:SEL on [page 173](#)), and be using AM demodulation or modulation (see :AFAN:INP on [page 102](#)).

### **Syntax**

:MEAS:AFR:DCAM <meas cmd> ! see "Number Measurement Syntax" on [page 193](#)

:MEAS:AFR:DCAM? !returns a real value

### **Example**

:MEAS:AFR:DCAM:AVERage 4 !averages DCAM over 4 measurements

### **Screen/field equivalent**

:AFR:DCAM reads the DC AM measurement field on the AF ANALYZER and RF ANALYZER screens.

## **:AFRequency:DCFM <measurement command>** **:AFRequency:DCFM?**

:AFR:DCAM controls/reads the DC FM measurement. To use this measurement, you must have the DC Level measurement selected (see :MEAS:AFR:SEL on [page 173](#)), and be using FM demodulation or modulation (see :AFAN:INP on [page 102](#)).

### **Syntax**

:MEAS:AFR:DCFM <meas cmd> ! see "Number Measurement Syntax" on [page 193](#)

:MEAS:AFR:DCFM? !returns a real value

### **Example**

:MEAS:AFR:DCFM:AVERage 4 !averages DCFM over 4 measurements

### **Screen/field equivalent**

:AFR:DCFM reads the DC FM measurement field on the AF ANALYZER and RF ANALYZER screens.

## **:AFRequency:DCVolts <measurement command>** **:AFRequency:DCVolts?**

:AFR:DCV controls/queries the DC Voltmeter. To use this measurement, you must have the DC Level measurement selected (see :MEAS:AFR:SEL on [page 173](#)), and have selected one of the following audio sources: SSB Demod, Audio In, Ext Mod, or Audio Out (see :AFAN:INP on [page 102](#)).

### **Syntax**

:MEAS:AFR:DCV <meas cmd> ! see "Number Measurement Syntax" on [page 193](#)

:MEAS:AFR:DCV? !returns a real value

### **Example**

:MEAS:AFR:DCV mV !sets the measurement unit to mV

:MEAS:AFR:DCV? !returns the real value in millivolts

### **Screen/field equivalent**

:AFR:DCV reads the DC Level measurement field on the AF ANALYZER and RF ANALYZER screens.

## **:AFRequency:DISToortion <measurement command>** **:AFRequency:DISToortion?**

:AFR:DIST controls/queries the Distortion measurement. This measurement is selected by using the :MEAS:AFR:SEL 'DISTN' command (see [page 173](#)), then you can read the measurement using the :AFR:DIST query command.

### **Syntax**

:MEAS:AFR:DIST <meas cmd> ! see "Number Measurement Syntax" on [page 193](#)

:MEAS:AFR:DIST? ! returns a real value

### **Example**

:MEAS:AFR:DEL 'DISTN' ! activates the Distortion measurement

:MEAS:AFR:DIST? !returns the measurement's value

### **Screen/field equivalent**

:AFR:DIST reads the Distortion measurement field on the AF ANALYZER and RF ANALYZER screens.

## **:AFRequency:FM <measurement command>** **:AFRequency:FM?**

:AFR:FM controls/queries the FM Deviation measurement. This measurement is active when FM Mod or FM Demod is selected by using the :AFAN:INP command (see [page 102](#)) and SNR is not selected by the :MEAS:AFR:SEL command (see [page 173](#)).

### **Syntax**

:MEAS:AFR:FM <meas cmd> ! see "Number Measurement Syntax" on [page 193](#)

:MEAS:AFR:FM? !returns real value

### **Example**

:MEAS:AFR:FM:AVERage:RESet !resets the number of averages

:MEAS:AFR:FM? !returns the measurement's value

### **Screen/field equivalent**

:AFR:FM reads the FM Deviation measurement field on the AF ANALYZER and RF ANALYZER screens.

## **:AFRequency:FREQuency <measurement command>** **:AFRequency:FREQuency?**

:AFR:FREQ controls/reads the Audio Frequency measurement. This measurement is active when AF Freq is selected by using the :MEAS:AFR:SEL 'AF Freq' command (see [page 173](#)).

### **Syntax**

:MEAS:AFR:FREQ <meas cmd> ! see "Number Measurement Syntax" on [page 193](#)

:MEAS:AFR:FREQ? ! returns real value

### **Example**

:MEAS:AFR:SEL 'AF Freq' !activates the measurement

:MEAS:AFR:FREQ? ! returns the measurement's value

### **Screen/field equivalent**

:AFR:FREQ reads the Audio Frequency measurement field on the AF ANALYZER and RF ANALYZER screens.

## **:AFRequency:SElect <measurement command>** **:AFRequency:SElect?**

:AFR:SEL controls/queries which measurement is displayed in the AF ANALYZER and RF ANALYZER screens.

### **Syntax**

:MEAS:AFR:SEL 'SINAD'

'Distn'

'SNR'

'AF Freq'

'DC Level'

:MEAS:AFR:SEL ! returns the presently selected measurement (string)

### **Example**

:MEAS:AFR:SEL AF Freq !displays the AF Frequency measurement

### **Screen/field equivalent**

:AFR:SEL corresponds to the measurement field in the AF ANALYZER and RF ANALYZER screens.

## **:AFRequency:SINAD <measurement command>** **:AFRequency:SINAD?**

:AFR:SINAD controls/queries the SINAD measurement. SINAD must first be selected using the :MEAS:AFR:SEL 'SINAD' command (see [page 173](#)).

### **Syntax**

:MEAS:AFR:SINAD <meas cmd> !controls the SINAD command

:MEAS:AFR:SINAD? !returns the measurement's value

### **Example**

:MEAS:AFR:SINAD !displays the measurement

### **Screen/field equivalent**

:AFR:SINAD reads the SINAD measurement field on the AF ANALYZER and RF ANALYZER screens.

## **:AFRequency:SNR <measurement command>** **:AFRequency:SNR?**

:AFR:SNR controls/queries the SNR measurement. SNR must first be selected using the :MEAS:AFR:SEL 'SNR' command (see [page 173](#)).

### **Syntax**

:MEAS:AFR:SNR <meas cmd> !controls the SNR command

:MEAS:AFR:SNR? !returns the measurement's value

### **Example**

:MEAS:AFR:SNR !displays the measurement

### **Screen/field equivalent**

:AFR:SNR corresponds to the SNR measurement field on the AF ANALYZER and RF ANALYZER screens.

## **:CANalyzer:ADC:APOWeradc:LEVel <measurement command>** **:CANalyzer:ADC:APOWeradc:LEVel?**

:ADC:APOW:LEV controls/queries the ADC for the Average Power measurement. The ADC is a measure of how close the actual input signal is to the maximum input level.

This measurement uses the Number Measurement Syntax, but does not use the Meter command from this syntax.

### **Syntax**

:MEAS:CAN:ADC:APOW:LEV <meas cmd> !see "[Number Measurement Syntax](#)" on [page 193](#)

:MEAS:CAN:ADC:APOW:LEV?

### **Example**

:MEAS:CAN:ADC:APOW:LEV? !displays the measurement

### **Screen/field equivalent**

:CAN:ADC:APOW:LEV corresponds to the ADC measurement field on the CDMA ANALYZER screen.



**:CANalyzer:ADC:RHOadc:LEVel <measurement command>**  
**:CANalyzer:ADC:RHOadc:LEVel?**

:ADC:RHO:LEV controls/queries the ADC for the Rho measurement. The ADC is a measure of how close the actual input signal is to the maximum input level.

This measurement uses the Number Measurement Syntax, but does not use the Meter command from this syntax.

**Syntax**

:MEAS:CAN:ADC:RHO:LEV <meas cmd> !see "[Number Measurement Syntax](#)" on page 193

:MEAS:CAN:ADC:RHO:LEV?

**Example**

:MEAS:CAN:ADC:RHO:LEV? !displays the measurement

**Screen/field equivalent**

:CAN:ADC:RHO:LEV corresponds to the ADC measurement field on the CDMA ANALYZER screen.

**:CANalyzer:ADC:EVMadc:LEVel <measurement command>**  
**:CANalyzer:ADC:EVMadc:LEVel?**

:ADC:EVM:LEV controls/queries the ADC for the EVM measurement. The ADC is a measure of how close the actual input signal is to the maximum input level.

This measurement uses the Number Measurement Syntax, but does not use the Meter command from this syntax.

**Syntax**

:MEAS:CAN:ADC:EVM:LEV <meas cmd> !see "[Number Measurement Syntax](#)" on page 193

:MEAS:CAN:ADC:EVM:LEV?

**Example**

:MEAS:CAN:ADC:EVM:LEV? !displays the measurement

**Screen/field equivalent**

:CAN:ADC:EVM:LEV corresponds to the ADC measurement field on the CDMA ANALYZER screen.

**:CANalyzer:ADC:TPOWeradc:LEVel <measurement command>**  
**:CANalyzer:ADC:TPOWeradc:LEVel?**

:ADC:TPOW:LEV controls/queries the ADC for the Channel Power measurement. The ADC is a measure of how close the actual input signal is to the maximum input level.

This measurement uses the Number Measurement Syntax, but does not use the Meter command from this syntax.

**Syntax**

:MEAS:CAN:ADC:TPOW:LEV <meas cmd> !see "[Number Measurement Syntax](#)" on page 193

:MEAS:CAN:ADC:TPOW:LEV?

**Example**

:MEAS:CAN:ADC:TPOW:LEV? !displays the measurement

**Screen/field equivalent**

:CAN:ADC:TPOW:LEV corresponds to the ADC measurement field on the CDMA ANALYZER screen.

**:CANalyzer:CARrier:FEEDthrough <measurement command>**  
**:CANalyzer:CARrier:FEEDthrough?**

:CAN:CAR:FEED controls/queries the Carrier Feedthrough measurement.

This measurement uses the Number Measurement Syntax, but does not use the Meter command from this syntax.

**Syntax**

:MEAS:CAN:CAR:FEED <meas cmd> !see "[Number Measurement Syntax](#)" on page 193

:MEAS:CAN:CAR:FEED? !returns the level

**Example**

:MEAS:CAN:CAR:FEED?

**Screen/field equivalent**

:CAN:CAR:FEED corresponds to the Carrier Feedthru measurement field on the CDMA ANALYZER screen.

**:CANalyzer:ERRor:FREQuency <measurement command>**  
**:CANalyzer:ERRor:FREQuency?**

:CAN:ERR:FREQ controls/queries the Frequency Error measurement.

This measurement uses the Number Measurement Syntax, but does not use the Meter command from this syntax.

**Syntax**

:MEAS:CAN:ERR:FREQ <meas cmd> !see ["Number Measurement Syntax" on page 193](#)

:MEAS:CAN:ERR:FREQ?

**Example**

:MEAS:CAN:ERR:FREQ?

**Screen/field equivalent**

:CAN:ERR:FREQ corresponds to the Frequency Error measurement field on the CDMA ANALYZER screen.

**:CANalyzer:ERRor:MAGNitude <measurement command>**  
**:CANalyzer:ERRor:MAGNitude?**

:CAN:ERR:MAGN controls/queries the Magnitude Error measurement.

This measurement uses the Number Measurement Syntax, but does not use the Meter command from this syntax.

**Syntax**

:MEAS:CAN:ERR:MAGN <meas cmd> !see ["Number Measurement Syntax" on page 193](#)

:MEAS:CAN:ERR:MAGN?

**Example**

:MEAS:CAN:ERR:MAGN?

**Screen/field equivalent**

:CAN:ERR:MAGN corresponds to the Error Magnitude measurement field on the CDMA ANALYZER screen.

## **:CANalyzer:ERRor:PHASe <measurement command>** **:CANalyzer:ERRor:PHASe?**

:CAN:ERR:PHAS controls/queries the Phase Error measurement.

This measurement uses the Number Measurement Syntax, but does not use the Meter command from this syntax.

### **Syntax**

:MEAS:CAN:ERR:PHAS <meas cmd> !see "[Number Measurement Syntax](#)" on page 193

:MEAS:CAN:ERR:PHAS?

### **Example**

:MEAS:CAN:ERR:PHAS?

### **Screen/field equivalent**

:CAN:ERR:PHAS corresponds to the Phase Error measurement field on the CDMA ANALYZER screen.

## **:CANalyzer:EVM <measurement command>** **:CANalyzer:EVM?**

:CAN:EVM controls/queries the Error Vector Magnitude (EVM) measurement.

This measurement uses the Number Measurement Syntax, but does not use the Meter command from this syntax.

### **Syntax**

:MEAS:CAN:EVM <meas cmd> !see "[Number Measurement Syntax](#)" on page 193

:MEAS:CAN:EVM?

### **Example**

:MEAS:CAN:EVM? !returns EVM in %

### **Screen/field equivalent**

:CAN:EVM corresponds to the EVM measurement field on the CDMA ANALYZER screen.

**:CANalyzer:POWer:AVG <measurement command>**  
**:CANalyzer:POWer:AVG?**

:CAN:POW:AVG control/queries the Average Power measurement.

This measurement uses the Number Measurement Syntax, but does not use the Meter command from this syntax.

**Syntax**

:MEAS:CAN:POW:AVG <meas cmd> !see "[Number Measurement Syntax](#)" on page 193

:MEAS:CAN:POW:AVG?

**Example**

:MEAS:CAN:POW:AVG? !returns the measurement

**Screen/field equivalent**

:CAN:POW:AVG corresponds to the Avg Pwr measurement field on the CDMA ANALYZER screen.

**:CANalyzer:POWer:CHANnel <measurement command>**  
**:CANalyzer:POWer:CHANnel?**

:CAN:POW:CHAN control/queries the Channel Power measurement.

This measurement uses the Number Measurement Syntax, but does not use the Meter command from this syntax.

**Syntax**

:MEAS:CAN:POW:CHAN <meas cmd> !see "[Number Measurement Syntax](#)" on page 193

:MEAS:CAN:POW:CHAN?

**Example**

:MEAS:CAN:POW:CHAN? !displays the measurement

**Screen/field equivalent**

:CAN:POW:CHAN corresponds to the Chan Pwr measurement field on the CDMA ANALYZER screen.

**:CANalyzer:RHO <measurement command>**  
**:CANalyzer:RHO?**

:CAN:RHO controls/queries the Rho measurement.

This measurement uses the Number Measurement Syntax, but does not use the Meter command from this syntax.

**Syntax**

:MEAS:CAN:RHO <meas cmd> !see "[Number Measurement Syntax](#)" on [page 193](#)

:MEAS:CAN:RHO?

**Example**

:MEAS:CAN:RHO?

**Screen/field equivalent**

:CAN:RHO corresponds to the Rho measurement field on the CDMA ANALYZER screen.

**:CANalyzer:TIME:OFFSet <measurement command>**  
**:CANalyzer:TIME:OFFSet?**

:CAN:RHO controls/queries the Time Offset measurement.

This measurement uses the Number Measurement Syntax, but does not use the Meter command from this syntax.

**Syntax**

:MEAS:CAN:TIME:OFFS <meas cmd> !see "[Number Measurement Syntax](#)" on [page 193](#)

:MEAS:CAN:TIME:OFFS?

**Example**

:MEAS:CAN:TIME:OFFS?

**Screen/field equivalent**

:CAN:TIMS:OFFS corresponds to the Time Ofs measurement field on the CDMA ANALYZER screen.

**:CDANalyzer:ADC <measurement command>**  
**:CDANalyzer:ADC?**

:CDAN:ADC controls/queries the ADC measurement in the Code Domain Analyzer.

This measurement uses the Number Measurement Syntax, but does not use the Meter command from this syntax.

**Syntax**

:MEAS:CDAN:ADC <meas cmd> !see ["Number Measurement Syntax" on page 193](#)

:MEAS:CDAN:ADC?

**Example**

:MEAS:CDAN:ADC?

**Screen/field equivalent**

:CDAN:ADC reads the ADCFs measurement field on the CODE DOMAIN ANALYZER screen.

**:CDANalyzer:CARRier:FEEDthrough <measurement command>**  
**:CDANalyzer:CARRier:FEEDthrough?**

:CDAN:CARR:FEED controls/queries the Carrier Feedthrough measurement on the Code Domain Analyzer.

This measurement uses the Number Measurement Syntax, but does not use the Meter command from this syntax.

**Syntax**

:MEAS:CDAN:CARR:FEED <meas cmd> !see ["Number Measurement Syntax" on page 193](#)

:MEAS:CDAN:CARR:FEED?

**Example**

:MEAS:CDAN:CARR:FEED?

**Screen/field equivalent**

:CDAN:CARR:FEED reads the Car FT measurement field on the CODE DOMAIN ANALYZER screen.

## **:CDANalyzer:ERRor:FREQuency <measurement command> :CDANalyzer:ERRor:FREQuency?**

:CDAN:CARR:FEED controls/queries the Carrier Feedthrough measurement on the Code Domain Analyzer.

This measurement uses the Number Measurement Syntax, but does not use the Meter command from this syntax.

### **Syntax**

:MEAS:CDAN:ERR:FREQ <meas cmd> !see ["Number Measurement Syntax" on page 193](#)

:MEAS:CDAN:ERR:FREQ?

### **Example**

:MEAS:CDAN:ERR:FREQ?

### **Screen/field equivalent**

:CDAN:ERR:FREQ reads the Freq Err measurement field on the CODE DOMAIN ANALYZER screen.

## **:CDANalyzer:MARKer:LEVel <measurement command> :CDANalyzer:MARKer:LEVel?**

:CDAN:MARK:LEV controls/queries the Marker Level of the Code Domain Analyzer. This measurement is made relative to the value at Walsh Code 0.

This measurement uses the Number Measurement Syntax, but does not use the Meter command from this syntax.

### **Syntax**

:MEAS:CDAN:MARK:LEV <meas cmd> !see ["Number Measurement Syntax" on page 193](#)

:MEAS:CDAN:MARK:LEV?

### **Example**

:MEAS:CDAN:MARK:LEV?

### **Screen/field equivalent**

:CDAN:MARK:LEV reads the Level field of the Marker on the CODE DOMAIN ANALYZER screen.



**:CDANalyzer:MARKer:PHASe <measurement command>**  
**:CDANalyzer:MARKer:PHASe?**

:CDAN:MARK:PHAS controls/queries the Marker Phase of the Code Domain Analyzer. This measurement is made relative to the value at Walsh Code 0.

This measurement uses the Number Measurement Syntax, but does not use the Meter command from this syntax.

**Syntax**

:MEAS:CDAN:MARK:PHAS <meas cmd> !see "[Number Measurement Syntax](#)" on page 193

:MEAS:CDAN:MARK:PHAS?

**Example**

:MEAS:CDAN:MARK:PHAS?

**Screen/field equivalent**

:CDAN:MARK:PHAS reads the Phase field of the Marker on the CODE DOMAIN ANALYZER screen.

**:CDANalyzer:MARKer:TIME <measurement command>**  
**:CDANalyzer:MARKer:TIME?**

:CDAN:MARK:TIME controls/queries the Marker Time of the Code Domain Analyzer. This measurement is made relative to the value at Walsh Code 0.

This measurement uses the Number Measurement Syntax, but does not use the Meter command from this syntax.

**Syntax**

:MEAS:CDAN:MARK:TIME <meas cmd> !see "[Number Measurement Syntax](#)" on page 193

:MEAS:CDAN:MARK:TIME?

**Example**

:MEAS:CDAN:MARK:PHAS?

**Screen/field equivalent**

:CDAN:MARK:PHAS reads the Time Offset field of the Marker on the CODE DOMAIN ANALYZER screen.

## **:CDANalyzer:TIME:OFFSet <measurement command>** **:CDANalyzer:TIME:OFFSet?**

:CDAN:TIME:OFFS controls/queries the Time Offset measurement of the Code Domain Analyzer.

This measurement uses the Number Measurement Syntax, but does not use the Meter command from this syntax.

### **Syntax**

:MEAS:CDAN:TIME:OFFS <meas cmd> !see "[Number Measurement Syntax](#)" on page 193

:MEAS:CDAN:TIME:OFFS?

### **Example**

:MEAS:CDAN:TIME:OFFS? !displays the measurement

### **Screen/field equivalent**

:CDAN:TIME:OFFS reads the Time Ofs measurement field on the CODE DOMAIN ANALYZER screen.

## **:CDANalyzer:TRACe:POWer?**

:CDAN:TRAC:POW controls/queries the power in the trace points in the Code Domain Analyzer. The measurement returns 64 values (in dB), one for each point.

This measurement uses the Number Measurement Syntax, but does not use the Meter command from this syntax.

### **Syntax**

:MEAS:CDAN:TRAC:POW <meas cmd> !see "[Number Measurement Syntax](#)" on page 193

:MEAS:CDAN:TRAC:POW?

### **Example**

:MEAS:CDAN:TRAC:POW?

### **Screen/field equivalent**

:CDAN:TRAC:POW does not correspond to any specific field in the CODE DOMAIN ANALYZER screen.

## **:CDANalyzer:TRACe:TOFFset?**

:CDAN:TRAC:TOFF controls/queries the timing in the trace points in the Code Domain Analyzer. The measurement returns 64 values (in seconds), one for each point.

This measurement uses the Number Measurement Syntax, but does not use the Meter command from this syntax.

### **Syntax**

:MEAS:CDAN:TRAC:TOFF <meas cmd> !see "Number Measurement Syntax" on page 193

:MEAS:CDAN:TRAC:TOFF?

### **Example**

:MEAS:CDAN:TRAC:TOFF?

### **Screen/field equivalent**

:CDAN:TRAC:TOFF does not correspond to any specific field in the CODE DOMAIN ANALYZER screen.

## **:CDANalyzer:TRACe:PERRor?**

:CDAN:TRAC:PERR controls/queries the phase in the trace points in the Code Domain Analyzer. The measurement returns 64 values (in radians), one for each point.

This measurement uses the Number Measurement Syntax, but does not use the Meter command from this syntax.

### **Syntax**

:MEAS:CDAN:TRAC:PERR <meas cmd> !see "Number Measurement Syntax" on page 193

:MEAS:CDAN:TRAC:PERR?

### **Example**

:MEAS:CDAN:TRAC:PERR?

### **Screen/field equivalent**

:CDAN:TRAC:PERR does not correspond to any specific field in the CODE DOMAIN ANALYZER screen.

**:OSCilloscope:MARKer:LEVel:AM <measurement command>**  
**:OSCilloscope:MARKer:LEVel:AM?**

:OSC:MARK:LEV:AM controls/queries the AM depth of modulation at the Oscilloscope's marker position.

This measurement uses the Number Measurement Syntax, but does not use the Meter command from this syntax.

**Syntax**

:MEAS:OSC:MARK:LEV:AM <meas cmd> !see "[Number Measurement Syntax](#)" on page 193

:MEAS:OSC:MARK:LEV:AM?

**Example**

:MEAS:OSC:MARK:LEV:AM?

**Screen/field equivalent**

:OSC:MARK:LEV:AM reads the Marker Lvl measurement field on the OSCILLOSCOPE screen.

**:OSCilloscope:MARKer:LEVel:FM <measurement command>**  
**:OSCilloscope:MARKer:LEVel:FM?**

:OSC:MARK:LEV:FM controls/queries the FM deviation at the Oscilloscope's marker position.

This measurement uses the Number Measurement Syntax, but does not use the Meter command from this syntax.

**Syntax**

:MEAS:OSC:MARK:LEV:FM <meas cmd> !see "[Number Measurement Syntax](#)" on page 193

:MEAS:OSC:MARK:LEV:FM?

**Example**

:MEAS:OSC:MARK:LEV:FM?

**Screen/field equivalent**

:OSC:MARK:LEV:FM reads the Marker Lvl measurement field on the OSCILLOSCOPE screen.

## **:OSCilloscope:MARKer:LEVel:VOLTs <measurement command>**

### **:OSCilloscope:MARKer:LEVel:VOLTs?**

:OSC:MARK:LEV:VOLT controls/queries the FM deviation at the Oscilloscope's marker position.

This measurement uses the Number Measurement Syntax, but does not use the Meter command from this syntax.

#### **Syntax**

:MEAS:OSC:MARK:LEV:VOLT <meas cmd> !see "Number Measurement Syntax" on page 193

:MEAS:OSC:MARK:LEV:VOLT?

#### **Example**

:MEAS:OSC:MARK:LEV:VOLT?

#### **Screen/field equivalent**

:OSC:MARK:LEV:VOLT reads the Marker Lvl measurement field on the OSCILLOSCOPE screen.

## **:OSCilloscope:MARKer:TIME <measurement command>**

### **:OSCilloscope:MARKer:TIME?**

:OSC:MARK:TIME controls/queries the time elapsed from the trigger event to the marker location.

This measurement uses the Number Measurement Syntax, but does not use the Meter command from this syntax.

#### **Syntax**

:MEAS:OSC:MARK:TIME <meas cmd> !see "Number Measurement Syntax" on page 193

:MEAS:OSC:MARK:TIME? !returns a real number

#### **Example**

:MEAS:OSC:MARK:TIME? !displays the measurement

#### **Screen/field equivalent**

:OSC:MARK:TIME corresponds to the Time marker field on the OSCILLOSCOPE screen.

## **:OSCilloscope:TRACe?**

:OSC:TRAC returns an array of 417 real values, corresponding to points on the Oscilloscope display. 0 corresponds to the first value (left side of trace display) and 416 is the last value (right side of the trace display).

### **Syntax**

:MEAS:OSC:TRAC?

### **Screen/field equivalent**

:OSC:TRAC has no corresponding field on the OSCILLOSCOPE screen.

## **:RFRequency:SElect <string> :RFRequency:SElect?**

:RFR:SEL selects/queries the RF measurements for the RF Analyzer.

### **Syntax**

:MEAS:RFR:SEL <meas cmd> !see "Number Measurement Syntax" on page 193

:MEAS:RFR:SEL?

### **Example**

:MEAS:RFR:SEL 'Frequency'

'Freq Error'

:MEAS:RFR:SEL?

### **Screen/field equivalent**

:RFR:SEL controls the measurement field on the RF ANALYZER screen.

## **:RFRequency:FREQuency:ABSolute <measurement command> :RFRequency:FREQuency:ABSolute?**

:RFR:FREQ:ABS controls/queries the RF frequency.

### **Syntax**

:RFR:FREQ:ABS <meas syntax> !see "Number Measurement Syntax" on page 193

:RFR:FREQ:ABS?

### **Screen/field equivalent**

:RFR:FREQ:ABS reads the Frequency measurement field on the RF ANALYZER screen.

**:RFRequency:FREQuency:ERRor <measurement command>**  
**:RFRequency:FREQuency:ERRor?**

:RFR:FREQ:ABS controls/queries the RF frequency error.

**Syntax**

:RFR:FREQ:ERR <meas syntax> !see "Number Measurement Syntax" on page 193

:RFR:FREQ:ERR?

**Screen/field equivalent**

:RFR:FREQ:ERR reads the Freq Error measurement field on the RF ANALYZER screen.

**:RFRequency:POWer <measurement command>**  
**:RFRequency:POWer?**

:RFR:POW controls/queries the transmitter power measurement. This is a peak power measurement.

**Syntax**

:MEAS:RFR:POW <meas cmd> !see "Number Measurement Syntax" on page 193

:MEAS:RFR:POW? !returns real value

**Example**

:MEAS:RFR:POW? !returns the value

**Screen/field equivalent**

:RFR:POW corresponds to the TX Power measurement field on the RF GENERATOR screen.

## **:SAnalyzer:MARKer:FREQuency <measurement command> :SAnalyzer:MARKer:FREQuency?**

:SAN:MARK:FREQ controls/queries the frequency at the current marker position of the Spectrum Analyzer.

This measurement uses the Number Measurement Syntax, but does not use the Meter command from this syntax.

### **Syntax**

:MEAS:SAN:MARK:FREQ <meas cmd> !see "[Number Measurement Syntax](#)" on page 193

:MEAS:SAN:MARK:FREQ? !returns real value

### **Example**

:MEAS:SAN:MARK:FREQ? !returns the value

### **Screen/field equivalent**

:SAN:MARK:FREQ reads the Freq Marker measurement field on the SPECTRUM ANALYZER screen.

## **:SAnalyzer:MARKer:LEVel <measurement command> :SAnalyzer:MARKer:LEVel?**

:SAN:MARK:LEV controls/queries the RF level at the current marker position of the Spectrum Analyzer.

This measurement uses the Number Measurement Syntax, but does not use the Meter command from this syntax.

### **Syntax**

:MEAS:SAN:MARK:LEV <meas cmd> !see "[Number Measurement Syntax](#)" on page 193

:MEAS:SAN:MARK:LEV? !returns real value

### **Example**

:MEAS:SAN:MARK:LEV? !returns the value

### **Screen/field equivalent**

:SAN:MARK:LEV reads the Lvl measurement field on the SPECTRUM ANALYZER screen.



## **:SAnalyzer:TRACe?**

:SAN:TRAC returns an array of 417 real values, corresponding to points on the Spectrum Analyzer display. 0 corresponds to the first value (left side of trace display) and 416 is the last value (right side of the trace display).

This measurement uses the Number Measurement Syntax, but does not use the Meter command from this syntax.

### **Syntax**

:MEAS:SAN:TRAC?

### **Example**

:MEAS:SAN:TRAC? !returns the array

### **Screen/field equivalent**

:SAN:TRAC has no corresponding field on the SPECTRUM ANALYZER screen.

## Multiple Real Number Setting Syntax

This section defines the syntax to be used with commands that require the Real Number Setting Syntax, but allow multiple numbers to be set in one command.

This format is typically used for entering values that are grouped in rows or columns.

The format is the same as for the Real Number Setting Syntax, except that each real number is preceded by an integer that identifies the relative position in the list of numbers.

### Syntax

*:Previous Syntax* <integer\_value>,<real\_value>[display unit\_of\_measure]

!integer\_value is the position, real\_value is the actual value

*:Previous Syntax?* <integer\_value> !returns the value of the position

*:Previous Syntax:*DUNits <integer\_value>,<display unit\_of\_measure>

!sets the units

*:Previous Syntax:*DUNits? <integer\_value> !returns the display units

*:Previous Syntax:*UNits <integer\_value>,<HP-IB unit\_of\_measure>

*:Previous Syntax:*UNits? <integer\_value> !Displays the HP-IB units

*:Previous Syntax:*INCRement <integer\_value>,<incr\_value>[display unit\_of\_measure]

!increments the present value

*:Previous Syntax:*INCRement <integer\_value>,UP | DOWN

*:Previous Syntax:*INCRement? <integer\_value> !returns the incr. value

*:Previous Syntax:*MODE <integer\_value>,LINear | LOGarithm

!sets mode of the command

*:Previous Syntax:*MODE? <integer\_value> !returns mode (LIN or LOG)

*:Previous Syntax:*MULTiPLY <integer\_value>!multiplies current setting

*:Previous Syntax:*DIVide <integer\_value>!divides current setting by 10

## Number Measurement Syntax

This syntax is used with measurement commands. It applies to both real and integer values.

This syntax is typically used with subsystems like the :MEASure subsystem/

### Syntax

*:Previous Syntax:*AUNits <HP-IB units> !sets the HP-IB units

*:Previous Syntax:*AUNits !returns the HP-IB units

*:Previous Syntax:*AVERage[:VALue] <real\_value>  
!sets the number of averages

*:Previous Syntax:*AVERage? !returns the number of averages

*:Previous Syntax:*AVERage:RESet !resets the number of averages to 1

*:Previous Syntax:*AVERage:STATe 1 | ON

*:Previous Syntax:*AVERage:STATe 0 | OFF

*:Previous Syntax:*AVERage:STATe? !returns 1 or 0

*:Previous Syntax:*DUNits <display unit\_of\_measure> !sets the disp. units

*:Previous Syntax:*DUNits? !returns the display units

*:Previous Syntax:*HLIMit[:VALue] <real\_value>[<units>]  
!sets the value of the High Limit for a measurement display

*:Previous Syntax:*HLIMit? !returns the High Limit setting

*:Previous Syntax:*HLIMit:DUNits <units> !display units

*:Previous Syntax:*HLIMit:DUNits? !returns the display units

*:Previous Syntax:*HLIMit:EXCeeded? !returns 1 or 0 to indicate if the High Limit was exceeded

*:Previous Syntax:*HLIMit:RESet !resets the limit

*:Previous Syntax:*HLIMit:STATe 1 | ON !sets state of the limit

*:Previous Syntax:*HLIMit:STATe 0 | OFF !sets state of the limit

*:Previous Syntax:*HLIMit:STATe? !returns 1 or 0

*:Previous Syntax:*LLIMit[:VALue] <real\_value>[<units>]  
!sets the value of the Low Limit for a measurement display

*:Previous Syntax:*LLIMit? !returns the Low Limit setting

*:Previous Syntax:LLIMit:DUNits <units> !display units*  
*:Previous Syntax:LLIMit:DUNits? !returns the display units*  
*:Previous Syntax:LLIMit:EXCeeded? !returns 1 or 0 to indicate if the  
Low  
Limit was exceeded*  
*:Previous Syntax:LLIMit:RESet !resets the limit*  
*:Previous Syntax:LLIMit:STATe 1 | ON !sets state of the limit*  
*:Previous Syntax:LLIMit:STATe 0 | OFF !sets state of the limit*  
*:Previous Syntax:LLIMit:STATe? !returns 1 or 0*  
*:Previous Syntax:METer[:STATe] 1 | ON*  
*:Previous Syntax:METer[:STATe] 0 | OFF*  
*:Previous Syntax:METer[:STATe]? !returns 1 or 0*  
*:Previous Syntax:METer:HEND | LEND <real value><units>*  
*:Previous Syntax:METer:HEND | LEND? !returns real value*  
*:Previous Syntax:METer:HEND | LEND:DUNits <units>*  
*:Previous Syntax:METer:HEND | LEND:DUNits?*  
*:Previous Syntax:METer:INTerval <integer value>*  
*:Previous Syntax:METer:INTerval? !returns integer value*  
*:Previous Syntax:REFerence[:VALue] <real\_value>[<units>]  
!sets the value of the reference for a measurement display*  
*:Previous Syntax:REFerence? !returns the reference setting*  
*:Previous Syntax:REFerence:DUNits <units> !display units*  
*:Previous Syntax:REFerence:DUNits? !returns the display units*  
*:Previous Syntax:REFerence:STATe 1 | ON !sets state of the reference*  
*:Previous Syntax:REFerence:STATe 0 | OFF !sets state of the reference*  
*:Previous Syntax:REFerence:STATe? !returns 1 or 0*  
*:Previous Syntax:STATe 1 | ON*  
*:Previous Syntax:STATe 0 | OFF*  
*:Previous Syntax:STATe? ! returns 1 or 0*  
*:Previous Syntax:UNits <HP-IB unit\_of\_measure> !sets the HP-IB  
units*  
*:Previous Syntax:UNits? !returns the HP-IB units*

### Examples

:RFAN:FREQ 850.35MHz !sets the frequency to 850.35 MHz

:RFAN:FREQ? !returns the frequency

:RFAN:FREQ:DUNits GHz !changes the units to GigaHertz

:RFAN:FREQ:INCR 3.5MHz !increments frequency by 3.5 MHz

:RFAN:FREQ:INCR? !returns the increment value

:RFAN:FREQ:MULT !multiplies the current frequency by 10

:RFAN:FREQ:MULT !multiplies the current frequency by 10

!Note that :STAT is not valid for :RFAN:FREQ

## **:OSCilloscope subsystem**

### **:CONTrol :CONTrol?**

:CONT selects/queries the Oscilloscope menus.

#### **Syntax**

:OSC:CONT 'Main'

    'Trigger'

    'Marker'

:OSC:CONT?

#### **Example**

:OSC:CONT 'Trigger' !displays the Trigger menu of the Oscilloscope

#### **Screen/field equivalent**

:CONT is used with the menu field of the OSCILLOSCOPE screen.

### **:MARKer:NPEak**

:MARK:NPE moves the marker to the minimum value of the average level of the display.

This command has no query.

#### **Syntax**

:OSC:MARK:NPE !moves the marker to the minimum

#### **Screen/field equivalent**

:OSC:MARK:NPE is used with the Marker To field of the OSCILLOSCOPE screen.

## **:MARKer:PPEak**

:MARK:PPE moves the marker to the maximum value of the average level of the display.

This command has no query.

### **Syntax**

:OSC:MARK:PPE !moves the marker to the maximum

### **Screen/field equivalent**

:OSC:MARK:PPE is used with the Marker To field of the OSCILLOSCOPE screen.

## **:MARKer:POSition <real number>**

### **:MARKer:POSition?**

:MARK:POS moves the marker according to the number of scale divisions from the left side of the screen.

This measurement uses the Real Number Setting Syntax, but does not use the :STATe command from this syntax

### **Syntax**

:OSC:MARK:POS <real number> !values 0 to 10.00

:OSC:MARL:POS? !returns the present position value

### **Example**

:OSC:MARK:POS 4.5 !positions the marker 4.5 divs from the left

### **Screen/field equivalent**

:OSC:MARK:POS is used with the Position To field of the OSCILLOSCOPE screen.

**:SCALE:TIME <string>**  
**:SCALE:TIME?**

:SCAL:TIME selects/queries the horizontal sweep time per division.

**Syntax**

:OSC:SCAL:TIME '200 ms'

'100 ms'

'50 ms'

'20 ms'

'10 ms'

'5 ms'

'2 ms'

'1 ms'

'500 us'

'200 us'

'100 us'

'50 us'

'20 us'

'10 us'

'5 us'

'2 us'

'1 us'

:OSC:SCAL:TIME? !returns present value

**Example**

:OSC:SCAL:TIME '2 ms' !sets scale to 2 ms

**Screen/field equivalent**

:OSC:SCL:TIME is used with the Time/Div field of the OSCILLOSCOPE screen.



**:SCALe:NADC <integer>**  
**:SCALe:NADC?**

This command uses the “[Integer Number Setting Syntax](#)”, without the :STAT command.

**Syntax**

:OSC:SCAL:NADC <integer>

:OSC:SCAL:NADC?

**Example**

:OSC:SCAL:NADC !displays the measurement

**Screen/field equivalent**

:SCAL:NADC is used with the menu field of the OSCILLOSCOPE screen.

## **:SCALE:VERTical:AM <string>** **:SCALE:VERTical:AM?**

:SCAL:VERT:AM sets/queries the vertical axis Amplitude per division when AM Modulation or AM Demodulation are selected in the AF Anl Input field (AF Analyzer screen).

### **Syntax**

:OSC:SCAL:VERT:AM '50%'

'20%'

'10%'

'5%'

'2%'

'1%'

'0.5%'

'0.2%'

'0.1%'

'0.05%'

:OSC:SCAL:VERT:AM?

### **Example**

:OSC:SCAL:VERT:AM '20%!'sets the vert scale

### **Screen/field equivalent**

:OSC:SCAL:VERT:AM is used with the Vert/Div field of the OSCILLOSCOPE screen.

**:SCALe:VERTical:FM <string>**  
**:SCALe:VERTical:FM?**

:SCAL:VERT:FM sets/queries the vertical axis Amplitude per division when FM Modulation or FM Demodulation are selected in the AF Anl Input field (AF Analyzer screen).

**Syntax**

:OSC:SCAL:VERT:FM '50 kHz'

'20 kHz'

'10 kHz'

'5 kHz'

'2 kHz'

'1 kHz'

'500 Hz'

'200 Hz'

'100 Hz'

'50 Hz'

'20 Hz'

'10 Hz'

:OSC:SCAL:VERT:FM?

**Example**

:OSC:SCAL:VERT:FM '20 kHz'

**Screen/field equivalent**

:SCAL:VERT:FM is used with the Vert/Div field of the OSCILLOSCOPE screen.

**:SCALE:VERTical:OFFSet <real number>**  
**:SCALE:VERTical:OFFSet?**

:SCAL:VERT:OFFS sets/queries the vertical scale (DC) offset of the vertical axis of the oscilloscope display. This moves the signal up to four divisions up or down with respect to the Oscilloscope's fixed center line.

This command uses the “[Real Number Setting Syntax](#)”, without the :STAT command.

**Syntax**

:OSC:SCAL:VERT:OFFS <real number>

:OSC:SCAL:VERT:OFFS?

**Example**

:OSC:SCAL:VERT:OFFS 2.5 !moves the signal 2.5 divisions

**Screen/field equivalent**

:OSC:SCAL:VERT:OFFS is used with the Vert Offset field of the OSCILLOSCOPE screen.

**:SCALe:VERTical:VOLTs <string>**  
**:SCALe:VERTical:VOLTs?**

:SCAL:VERT:VOLT sets/queries the vertical scale of the Oscilloscope display when Audio In is selected in the AF Anl screen. This command is valid when the AF Anl Input field in the AF Anl screen is set to anything except FM Mod, FM Demod, AM Mod, or AM Demod.

**Syntax**

:OSC:SCAL:VERT:VOLT '20 V'

'10 V'

'5 V'

'2 V'

'1 V'

'500 mv'

'200 mv'

'100 mv'

'50 mv'

'20 mv'

'10 mv'

'5 mv'

'2 mv'

'1 mv'

'500 uv'

'200 uv'

'100 uv'

'50 uv'

'20 uv'

:OSC:SCAL:VERT:VOLT?

**Example**

:SCAL:VERT:VOLT '5 V'!sets scal to 5V per division

**Screen/field equivalent**

:SCAL:VERT:VOLT is used with the Vert/Div field of the OSCILLOSCOPE screen.

## **:TRIGger:LEVel <real number>** **:TRIGger:LEVel?**

:TRIG:LEV sets/queries the level of the Oscilloscope's trigger.

This command uses the “[Real Number Setting Syntax](#)”, without the :STAT, :DUN, :UNIT, :INCR:MODE, :INCR:DUN, commands.

### **Syntax**

:OSC:TRIG:LEV <real> !values

:OSC:TRIG:LEV? !returns the level

### **Example**

:OSC:TRIG:LEV 0.4 !sets the level to 0.4 V

### **Screen/field equivalent**

:TRIG:LEV is used with the Level field of the Trigger in the OSCILLOSCOPE screen.

## **:TRIGger:MODE <string>** **:TRIGger:MODE?**

:TRIG:MODE sets/queries the retriggering mode of the trigger for the Oscilloscope. If “Cont” is chosen, then the oscilloscope is triggering continuously and is ready for another measurement after one has finished. If “Single” is chosen then the oscilloscope requires a trigger command before making another measurement. :OSC:RESet is the trigger command for the Oscilloscope.

### **Syntax**

:OSC:TRIG:MODE 'Cont'

'Single'

:OSC:TRIG:MODE?

### **Example**

:OSC:TRIG:MODE 'Single' !sets mode to single trigger

### **Screen/field equivalent**

:TRIG:MODE is used with the Trigger field of the OSCILLOSCOPE screen.

**:TRIGger:DELay <real number>**  
**:TRIGger:DELay?**

:TRIG:DEL sets/queries the trigger delay. Positive values delay the trigger, negative values apply a pre-trigger function to each measurement.

This command uses the “[Real Number Setting Syntax](#)”, without the :STAT, :DUN, :UNIT, :INCR:MODE, :INCR:DUN, commands.

The valid range of the delay depends on the Time/Div setting. See :OSC:SCAL:TIME to set the Time/Div.

**Syntax**

:OSC:TRIG:DEL <real> !values depend on Time/Div

:OSC:TRIG:DEL?

**Example**

:OSC:TRIG:DEL 0.2 !sets to 0.2 units

**Screen/field equivalent**

:TRIG:DEL is used with the Trig Delay field of the OSCILLOSCOPE screen.

**:TRIGger:PRETrigger <real number>**  
**:TRIGger:PRETrigger?**

:TRIG:PRET sets/queries a pretrigger for each measurement.

This command uses the “[Real Number Setting Syntax](#)”, without the :STAT, :DUN, :UNIT, :INCR:MODE, :INCR:DUN, commands.

**Syntax**

:OSC:TRIG:PRET <real>

:OSC:TRIG:PRET?

**Screen/field equivalent**

:TRIG:PRET has no equivalent field in the OSCILLOSCOPE screen.

## **:TRIGger:RESet**

:RESet triggers a measurement in the Oscilloscope.

### **Syntax**

:OSC:TRIG:RES !triggers the Oscilloscope

### **Screen/field equivalent**

:TRIG:RESet is used with the Reset field of the OSCILLOSCOPE screen.

## **:TRIGger:SENSe <string>**

### **:TRIGger:SENSe?**

:TRIG:SENS sets/queries the desired edge of the trigger. 'Pos' triggers the measurement on the positive-going edge of the input signal. 'Neg' triggers on the negative-going edge.

### **Syntax**

:OSC:TRIG:SENS 'Pos'

'Neg'

:OSC:TRIG:SENS?

### **Screen/field equivalent**

:TRIG:SENS is used with the Pos/Neg field of the OSCILLOSCOPE screen.

## **:TRIGger:SOURce <string>**

### **:TRIGger:SOURce?**

:TRIG:SOUR sets/queries the source of the trigger.

### **Syntax**

:OSC:TRIG:SOUR 'Internal'

'Ext(TTL)'

:OSC:TRIG:SOUR?

### **Screen/field equivalent**

:TRIG:SOUR is used with the Internal field of the OSCILLOSCOPE screen.



**:TRIGger:TYPE <string>**  
**:TRIGger:TYPE?**

:TRIG:TYPE sets/queries how the trigger level is set. 'Auto' triggers a measurement if a triggering signal is not detected within approximately 50 ms of the last trigger. 'Norm' requires a specific triggering signal before triggering.

**Syntax**

:OSC:TRIG:TYPE 'Auto'

          'Norm'

:OSC:TRIG:TYPE?

**Screen/field equivalent**

:TRIG:TYPE is used with the Auto/Norm field of the OSCILLOSCOPE screen.

## **:PROGram subsystem**

The Program subsystem provides a set of commands which allow an external controller to generate and control an IBASIC program within the Test Set.

These commands have no equivalent fields or screens.

### **[[:SElected]:DEFine <program data> [:SElected]:DEFine?**

:SEL:DEF downloads an IBASIC program into the Test Set. The query form returns the program. The program must be transferred as IEEE 488.2 Arbitrary Block Program Data. Refer to the IEEE standard 488.2-1987 for detailed information on this data type.

[[:SEL]] is an optional portion of this command.

<program> is the actual program content

#### **Syntax**

:PROG:SEL:DEF <#0><program><NL><END>

:PROG:DEF <#0><program><NL><END>

### **[[:SElected]:EXECute**

:SEL:EXEC executes (from an IBASIC controller) an IBASIC command in the Test Set's built-in IBASIC controller.

#### **Syntax**

:PROG:SEL:EXEC

:PROG:EXEC !equivalent command

### **[[:SElected]:STATe <variable> [:SElected]:STATe?**

:STAT sets/queries (from an external IBASIC controller) the execution state of the IBASIC program currently loaded in the Test Set.

#### **Syntax**

:PROG:SEL:STAT

:PROG:STAT !equivalent command

**[[:SElected]:NUMBer <variable>(,<nvalues>)  
[:SElected]:NUMBer?**

:SEL:NUMB sets/queries the value of numeric variables or arrays in the IBASIC program currently loaded in the Test Set.

**Syntax**

:PROG:SEL:NUMB <variable>,<nvalues>

:PROG:NUMB <variable>,<nvalues> !equivalent command

**[[:SElected]:STRing <variable>(,<nvalues>)  
[:SElected]:STRing?**

:SEL:STR sets/queries the value of string variables or arrays in the IBASIC program currently loaded in the Test Set

**Syntax**

:PROG:SEL:STR <variable>,<nvalues>

:PROG:STR <variable>,<nvalues> !equivalent command

**[[:SElected]:WAIT  
[:SElected]:WAIT?**

**Syntax**

:PROG:SEL:WAIT? !returns an integer value

**[[:SElected]:DELeTe  
[:SElected]:DELeTe:ALL**

:SEL:DEL deletes the IBASIC program currently loaded in the Test Set.

**Syntax**

:PROG:SEL:DEL

:PROG:DEL !equivalent command

:PROG:DEL:ALL !equivalent command

## Real Number Setting Syntax

This syntax is for setting real values in commands. It is to be used with commands that require the Real Number Setting Syntax.

An example of a command that requires the Real Number Setting Syntax is the :RFANalyzer:FREQUENCY command (see ":RFANalyzer subsystem" on page 215).

### Syntax

*:Previous Syntax* <real\_value>[display unit\_of\_measure]  
*:Previous Syntax?* !returns the value  
*:Previous Syntax*:DUNits <display unit\_of\_measure> !sets the units  
*:Previous Syntax*:DUNits? !returns the display units  
*:Previous Syntax*:UNits <HP-IB unit\_of\_measure>  
*:Previous Syntax*:UNits? !Displays the HP-IB units  
*:Previous Syntax*:INCRement <incr\_value>[display unit\_of\_measure]  
!increments the present value  
*:Previous Syntax*:INCRement? !returns the increment value  
*:Previous Syntax*:INCRement UP | DOWN !increments up or down  
*:Previous Syntax*:STATe 1 | ON  
*:Previous Syntax*:STATe 0 | OFF  
*:Previous Syntax*:STATe? ! returns 1 or 0  
*:Previous Syntax*:MODE LINear | LOGarithm !sets mode of the command  
*:Previous Syntax*:MODE? !returns mode (LIN or LOG)  
*:Previous Syntax*:MULTiply !multiplies current setting by 10  
*:Previous Syntax*:DIVide !divides current setting by 10

#### Examples

:RFAN:FREQ 850.35MHz !sets the frequency to 850.35 MHz  
:RFAN:FREQ? !returns the frequency  
:RFAN:FREQ:DUNits GHz !changes the units to GigaHertz  
:RFAN:FREQ:INCR 3.5MHz !increments frequency by 3.5 MHz  
:RFAN:FREQ:INCR? !returns the increment value

**:RFAN:FREQ:MULT !multiplies the current frequency by 10**

**:RFAN:FREQ:MULT !multiplies the current frequency by 10**

**!Note that :STAT is not valid for :RFAN:FREQ**

## **[:REGister] subsystem**

Register Subsystem contains the Save/Recall commands used in the Save/Recall registers.

### **:CLEar <integer or string>**

:CLE clears the register defined by the integer value or string argument.

#### **Syntax**

:CLE '<string>' !clears the register named 'string'

:CLE '<integer value>' !clears the numbered register

:REG:CLE '<string>' !equivalent command

#### **Screen/field equivalent**

This command corresponds to the Clear command in the Save/Recall menu, available in most screens.

### **:CLEar:ALL**

CLE clears all registers.

#### **Syntax**

:CLE:ALL

:REG:CLE:ALL !equivalent command

#### **Screen/field equivalent**

This command corresponds to the Clear All command in the Save/Recall menu, available in most screens.

## **:RECall <integer or string>**

:REC recalls the register defined by the integer value or string argument

### **Syntax**

:REC '<string>' !recalls the register named 'string'

:REC '<integer value>' !recalls the numbered register

:REG:REC '<string>' !equivalent command

### **Screen/field equivalent**

This command corresponds to the Recall command in the Save/Recall menu, available in most screens.

## **:RECall:LIST?**

:REC:LIST? lists the Save/Recall registers that have been named.

### **Syntax**

:REG:REC:LIST?

:REC:LIST? !equivalent command

### **Screen/field equivalent**

:REC:LIST? has no equivalent field.

## **:SAVE <integer or string>**

:SAVE saves the register defined by the integer value or string argument

### **Syntax**

:SAVE '<string>' !recalls the register named 'string'

:SAVE '<integer value>' !recalls the numbered register

:REG:SAVE '<string>' !equivalent command

### **Screen/field equivalent**

This command corresponds to the Save command in the Save/Recall menu, available in most screens.

## **:SAVE:LIST?**

:SAVE:LIST? lists the Save/Recall registers that have been named

### **Syntax**

:REG:SAVE:LIST?

:SAVE:LIST? !equivalent command

### **Screen/field equivalent**

:SAVE:LIST? has no equivalent field.



---

## **:RFANalyzer subsystem**

### **:ATTenuator <string>**

#### **:ATTenuator?**

:ATT sets/queries the amount of input attenuation in the path of the selected input port. :CONF:ATT:MODE 'Hold' overrides this command.

#### **Syntax**

:RFAN:ATT '40 dB'

'20 dB'

'0 dB'

RFAN:ATT?

#### **Screen/field equivalent**

:ATT corresponds to the Input Atten field on the INST CONFIGURE, RF ANALYZER, and SPECTRUM ANALYZER screens.

### **:ATTenuator:MODE <string>**

#### **:ATTenuator:MODE?**

:ATT:MODE sets/queries the RF Auto-ranging mode.  
:CONF:ATT:MODE 'Hold' overrides this command.

#### **Syntax**

:RFAN:ATT:MODE 'Auto'

'Hold'

:RFAN:ATT:MODE?

#### **Screen/field equivalent**

:ATT:MODE corresponds to the Input Atten field on the INST CONFIGURE, RF ANALYZER, and SPECTRUM ANALYZER screens.

## **:FREQuency <real number> :FREQuency?**

:FREQ sets/queries the tune frequency for the RF Analyzer. This command requires that the RF Display field be set to FREQ (Configure screen).

This command uses the “[Real Number Setting Syntax](#)”, without the :STAT command.

### **Syntax**

:RFAN:FREQ <real>

:RFAN:FREQ?

### **Screen/field equivalent**

:FREQ corresponds to the Tune Freq field on the RF ANALYZER screen.

## **:FREQuency:GTIMe <real number>**

:FREQ:GTIM sets/queries the gate time for the RF Frequency counter.

This command uses the “[Real Number Setting Syntax](#)”, but only includes the :DUNits and UNITs commands.

### **Syntax**

:RFAN:FREQ:GTIM <real>

:RFAN:FREQ:GTIM?

### **Screen/field equivalent**

:FREQ:GTIM has no equivalent field.

## **:GTIMe <real number> :GTIMe?**

:GTIM sets/queries the gate time for the RF Frequency counter.

This command uses the “[Real Number Setting Syntax](#)”, and only includes the :DUNits and UNITs commands.

### **Syntax**

:RFAN:GTIM <real>

:RFAN:GTIM?

### **Screen/field equivalent**

:FREQ:GTIM has no equivalent field.

**:IFBW <string>**  
**:IFBW?**

:IFBW sets/queries the IF filter bandwidth.

**Syntax**

:RFAN:IFBW '15 kHz'

'230 kHz'

:RFAN:IFBW?

**Screen/field equivalent**

:IFBW controls the IF Filter field on the RF ANALYZER screen.

**:INPut <string>**  
**:INPut?**

:INPut sets/queries the RF Input Port.

Note: Some measurements can only be made on one port.

**Syntax**

:RFAN:INP 'RF In'

'Ant'

:RFAN:INP?

**Screen/field equivalent**

:INP controls the RF Input Port field on the INST CONFIGURE, RF ANALYZER and CDMA ANALYZER screens.

**:PMEasurement:DETECTOR <string>**  
**:PMEasurement:DETECTOR?**

:PME:DET sets/queries the Analog TX power measurement method.

**Syntax**

:RFAN:PME:DET 'Peak'

'Sample'

:RFAN:PME:DET?

**Screen/field equivalent**

:PME:DET controls the TX PWR Meas field on the RF ANALYZER screen.

## **:PMEasurement:ZERO**

This command zeroes the TX power measurement.

### **Syntax**

:RFAN:PME:ZERO

### **Screen/field equivalent**

:PME:ZERO is equivalent to the TX Pwr Zero field on the RF ANALYZER screen.

## **:SENSitivity <string>**

### **:SENSitivity?**

:SENS sets/queries the RF Input sensitivity.

### **Syntax**

:RFAN:SENS 'Normal'

'High'

:RFAN:SENS?

### **Screen/field equivalent**

:SENS controls the Sensitivity field on the RF ANALYZER screen.

## **:SQUelch <string>**

### **:SQUelch?**

:SQU sets/queries the Squelch control setting.

### **Syntax**

:RFAN:SQU 'Pot'

'Open'

'Fixed'

:RFAN:SQU?

### **Screen/field equivalent**

:SQU controls the Squelch field on the RF ANALYZER screen.

---

## **:RFGenerator subsystem**

The RF Generator subsystem controls the functions of the RF GENERATOR screen.

### **:ATTenuator <string>**

#### **:ATTenuator?**

:ATT sets/queries the Attenuator Hold function.

#### **Syntax**

:RFG:ATT 'On'

'Off'

:RFG:ATT?

#### **Screen/field equivalent**

:ATT controls the Atten Hold field on the RF GENERATOR screen.

### **:AMPLitude <real number>**

:AMPL sets/queries the amplitude of the RF generator.

This field also controls the RF generator viewed in the Spectrum Analyzer.

This command uses the [“Real Number Setting Syntax”](#).

#### **Syntax**

:RFG:AMPL <real>

:RFG:AMPL?

#### **Screen/field equivalent**

:AMPL controls the Amplitude field on the RF GENERATOR screen.

## **:FM:COUpling <string>** **:FM:COUpling?**

:FM:COUP sets/queries the coupling between the Modulation Input port and the RF Generator's FM modulator.

### **Syntax**

:RFG:FM:COUP 'AC'  
          'DC'

:RFG:FM:COUP?

### **Screen/field equivalent**

:FM:COUP controls the FM Coupling field on the RF GENERATOR screen.

## **:FM:DCZero**

:FM:DCZ zeroes any DC bias that exists when DC is the chosen FM coupling.

### **Syntax**

:RFG:FM:DCZ

### **Screen/field equivalent**

:FM:DCZ controls the DC FM Zero field on the RF GENERATOR screen.

## **:FREQuency <real number>**

:FREQ sets/queries the frequency of the RF Generator. The Test Set must first be in FREQ mode (see ":CONFigure subsystem" on page 137 for the RF Display field).

This command uses the "Real Number Setting Syntax", without the :STAT command.

### **Syntax**

:RFG:FREQ <real number>  
:RFG:FREQ?

### **Screen/field equivalent**

:FREQ controls the Tune Freq field on the RF GENERATOR screen, when in FREQ mode.

**:MODulation:AOUT <string>**  
**:MODulation:AOUT?**

:MOD:AOUT sets/queries the coupling between the demodulated audio and the Audio Out Port.

**Syntax**

:RFG:MOD:AOUT 'AC'  
                  'DC'

:RFG:MOD:AOUT?

**Screen/field equivalent**

:MOD:AOUT controls the Audio Out field on the RF GENERATOR screen.

**:MODulation:EXTernal:AM <real number>**  
**:MODulation:EXTernal:AM?**

:MOD:EXT:AM sets/queries the AM sensitivity of the RF Generator when AM is applied through the modulation input port. The modulation input must be set to AM.

This command uses the [“Real Number Setting Syntax”](#).

**Syntax**

:MOD:EXT:AM <real number>  
:MOD:EXT:AM?

**Screen/field equivalent**

:MOD:EXT:AM controls the Mod In To field on the RF GENERATOR screen when AM has been selected in that same field.

## **:MODulation:EXTernal:DESTination <string> :MODulation:EXTernal:DESTination?**

:MOD:EXT:DEST sets/queries the type of modulation applied to the RF Generator.

This command works with :MOD:EXT:AM and :MOD:EXT:FM to set the level of modulation used.

### **Syntax**

:MOD:EXT:DEST 'AM (/Vpk)'

'FM (/Vpk)'

:MOD:EXT:DEST?

### **Screen/field equivalent**

:MOD:DEST:EXT controls the Mod In To field on the RF GENERATOR screen.

## **:MODulation:EXTernal:FM <real number> :MODulation:EXTernal:FM?**

:MOD:EXT:FM sets/queries the FM deviation of the RF Generator when FM is applied through the modulation input port. The modulation input must be set to FM.

This command uses the [“Real Number Setting Syntax”](#).

### **Syntax**

:MOD:EXT:FM <real number>

:MOD:EXT:FM?

### **Screen/field equivalent**

:MOD:EXT:FM controls the Mod In To field on the RF GENERATOR screen when FM has been selected in that same field..



**:OUTPut <string>**  
**:OUTPut?**

:OUTP sets/queries the RF Output Port.

**Syntax**

:RFG:OUTP 'RF Out'

'Dupl'

:RFG:OUTP?

**Screen/field equivalent**

:OUTP controls the Output Port field on the RF GENERATOR and SPECTRUM ANALYZER screens.

## **:SAnalyzer subsystem**

### **:ATTenuator <string>**

#### **:ATTenuator?**

:ATT sets/queries the attenuator setting for the input port.

#### **Syntax**

:SAN:ATT '40 dB'

'20 dB'

'0 dB'

:SAN:ATT?

#### **Screen/field equivalent**

:ATT controls the Input Atten field on the SPECTRUM ANALYZER screen.

### **:ATTenuator:MODE <string>**

#### **:ATTenuator:MODE?**

:ATT:MODE sets/queries the mode of the attenuator.

#### **Syntax**

:SAN:ATT:MODE 'Auto'

'Hold'

:SAN:ATT:MODE?

#### **Screen/field equivalent**

:ATT:MODE controls the Input Atten field on the SPECTRUM ANALYZER screen.

**:CFRequency <real number>**  
**:CFRequency?**

:CFR sets/queries the center frequency of the spectrum analyzer's display.

This command uses the "Real Number Setting Syntax", without the :STAT command.

**Syntax**

:SAN:CFR <real number>

:SAN:CFR?

**Screen/field equivalent**

:CFR controls the Center Freq field on the SPECTRUM ANALYZER screen.

**:CONTRol <string>**  
**:CONTRol?**

:CONT sets/queries the primary menu for the spectrum analyzer. From this menu you can choose controls for the RF generator, marker, or trigger.

**Syntax**

:SAN:CONT 'Main'

'RF Gen'

'Marker'

'Auxiliary'

:SAN:CONT?

**Screen/field equivalent**

:CONT controls the Controls field on the SPECTRUM ANALYZER screen.

## **:DISPlay:SCALe <string>** **:DISPlay:SCALe**

:DISP:SCAL sets/queries the vertical scale of the spectrum analyzer display.

### **Syntax**

:SAN:DISP:SCAL '1 dB/div'

'2 dB/div'

'10 dB/div'

:SAN:DISP:SCAL?

### **Screen/field equivalent**

:DISP:SCAL controls the scale field on the SPECTRUM ANALYZER screen.

## **:INPut <string>** **:INPut**

:INP sets/queries the input to the Spectrum Analyzer.

### **Syntax**

:SAN:INP 'RF In'

'Dupl'

:SAN:INP?

### **Screen/field equivalent**

:INP controls the Input field on Main control menu of the SPECTRUM ANALYZER screen.

## **:MARKer:CFRequency**

:MARK:CFR moves the marker to the center frequency of the spectrum analyzer.

### **Syntax**

:SAN:MARK:CFR

:SAN:MARK:CFR?

### **Screen/field equivalent**

:MARK:CFR controls the Marker To field on the Marker menu of the SPECTRUM ANALYZER screen.

## **:MARKer:NPEak**

:MARK:NPE moves the marker to the next peak of the spectrum analyzer display.

### **Syntax**

:SAN:MARK:NPE  
:SAN:MARK:NPE?

### **Screen/field equivalent**

:MARK:NPE controls the Marker To field on the Marker menu of the SPECTRUM ANALYZER screen.

## **:MARKer:EXCursion <integer number>** **:MARKer:EXCursion?**

:MARK:EXC increments or decrements the marker position.  
This command uses the [“Integer Number Setting Syntax”](#).

### **Syntax**

:SAN:MARK:EXC <integer>  
:SAN:MARK:EXC?

### **Screen/field equivalent**

:MARK:EXC controls the Position field on the SPECTRUM ANALYZER screen.

## **:MARKer:NPLLevel <real number>** **:MARKer:NPLLevel?**

:MARK:NPL is the level of the next peak signal.  
This command uses the [“Real Number Setting Syntax”](#), without the :STAT command.

### **Syntax**

:SAN:MARK:NPL <real number>  
:SAN:MARK:NPL?

### **Screen/field equivalent**

:MARK:NPL controls the Level field on the SPECTRUM ANALYZER screen.

## **:MARKer:PEAK**

:MARK:PEAK moves the marker to the highest peak of the signals on the spectrum analyzer display.

### **Syntax**

:SAN:MARK:PEAK

### **Screen/field equivalent**

:MARK:PEAK controls the Peak field in the Marker menu of the SPECTRUM ANALYZER screen.

## **:MARKer:POSition <real number> :MARKer:POSition?**

:MARK:POS sets/queries the position of the marker on the spectrum analyzer.

This command uses the “[Real Number Setting Syntax](#)”, without the :STAT command.

### **Syntax**

:SAN:MARK:POS <real number>

:SAN:MARK:POS?

### **Screen/field equivalent**

:MARK:POS controls the Position field in the Marker menu of the SPECTRUM ANALYZER screen.

## **:MARKer:RLEVEL**

:MARK:RLEV moves the marker to the reference level position of the spectrum analyzer.

### **Syntax**

:SAN:MARK:RLEV

### **Screen/field equivalent**

:MARK:RLEV controls the Marker To field in the marker menu of the SPECTRUM ANALYZER screen.

**:RFGenerator <string>**  
**:RFGenerator?**

:RFG selects between the fixed frequency generator and the tracking generator.

**Syntax**

:SAN:RFG 'Track'  
    'Fixed'  
:SAN:RFG?

**Screen/field equivalent**

:RFG controls an unnamed field in the RF Gen menu of the SPECTRUM ANALYZER screen.

**:RLEVel <real number>**  
**:RLEVel?**

:RLEV sets/queries the reference level of the spectrum analyzer.

This command uses the “[Real Number Setting Syntax](#)”, without the :STAT command.

**Syntax**

:SAN:RLEV <real number>  
:SAN:RLEV?

**Screen/field equivalent**

:RLEV controls the Ref Level field on the SPECTRUM ANALYZER screen.

**:SPAN <real number>**  
**:SPAN?**

:SPAN sets/queries the span of the spectrum analyzer display.

This command uses the “[Real Number Setting Syntax](#)”, without the :STAT command.

**Syntax**

:SAN:SPAN <real number>  
:SAN:SPAN?

**Screen/field equivalent**

:SPAN controls the Span field on the SPECTRUM ANALYZER screen.

**:TGENerator:AMPLitude <real number>**  
**:TGENerator:AMPLitude?**

:TGEN:AMPL sets/queries the amplitude of the tracking generator.  
This command uses the “[Real Number Setting Syntax](#)”.

**Syntax**

:SAN:TGEN:AMPL <real>  
:SAN:TGEN:AMPL?

**Screen/field equivalent**

:TGEN:AMPL controls the Amplitude field of the Tracking Generator of the SPECTRUM ANALYZER screen.

**:TGENerator:DESTination <string>**  
**:TGENerator:DESTination?**

:TGEN:DEST sets the output port for the tracking generator.

**Syntax**

:SAN:TGEN:DEST 'RF Out'  
'Dupl'  
:SAN:TGEN:DEST?

**Screen/field equivalent**

:TGEN:DEST controls the Port field on the SPECTRUM ANALYZER screen.

**:TGENerator:OFRequency <real number>**  
**:TGENerator:OFRequency?**

:TGEN:OFR sets/queries the offset frequency of the tracking generator.  
This command uses the “[Real Number Setting Syntax](#)”, without the :STAT command.

**Syntax**

:SAN:TGEN:OFR <real number>  
:SAN:TGEN:OFR?

**Screen/field equivalent**

:TGEN:OFR controls the Offset field in the Tracking Generator menu of the SPECTRUM ANALYZER screen.



**:TGENerator:SWEep <string>**  
**:TGENerator:SWEep?**

:TGEN:SWE set/queries the sweep type of the tracking generator.

**Syntax**

:SAN:TGEN:SWE 'Norm'

'Invert'

:SAN:TGEN:SWE?

**Screen/field equivalent**

:TGEN:SWE controls the Sweep field in the Tracking Generator menu of the SPECTRUM ANALYZER screen.

**:TRACe:MHOLd <string>**  
**:TRACe:MHOLd?**

:TRAC:MHOL sets/queries the type of averaging used in the spectrum analyzer's display.

**Syntax**

:SAN:TRAC:MHOL 'No Pk/Avg'

'Pk Hold'

'Avg 1'

'Avg 2'

'Avg 3'

'Avg 4'

'Avg 5'

'Avg 10'

'Avg 20'

'Avg 50'

'Avg 100'

'Off'

:SAN:TRAC:MHOL?

**Screen/field equivalent**

:TRAC:MHOL controls an unnamed field on the SPECTRUM ANALYZER screen.

**:TRACe:NORMAlize <string>**  
**:TRACe:NORMAlize?**

:TRAC:NORM sets/queries the type of display.

**Syntax**

:SAN:TRAC:NORM 'A Only'

'A-B'

:SAN:TRAC:NORM?

**Screen/field equivalent**

:TRAC:NORM controls the Normalize field in the tracking generator of the SPECTRUM ANALYZER screen.

**:TRACe:SAVE**

:TRAC:SAVE stores the present trace on the spectrum analyzer display.

**Syntax**

:SAN:TRAC:SAVE

**Screen/field equivalent**

:TRAC:SAVE controls the Save field on the SPECTRUM ANALYZER screen.

---

## :STATus

The STATus subsystem controls many of the modes of the Test Set, including the states. These commands do not correspond to fields in specific screens.

### :PRESet

:PRES presets the Test Set.

#### Syntax

:STAT:PRES

### :CALibration:CONDition?

:CAL:COND returns the state of the Calibration.

This query returns an integer value.

#### Syntax

:STAT:CAL:COND?

### :CALibration:ENABLE :CALibration:ENABLE?

:CAL:ENAB enables the calibration.

This query returns an integer value.

#### Syntax

:STAT:CAL:ENAB

:STAT:CAL:ENAB

### :CALibration[:EVENT]?

:CAL:EVEN returns the state of the event.

#### Syntax

:STAT:CAL?

**:CALibration:NTRansition**  
**:CALibration:NTRansition?**

:CAL:NTR returns the state of the negative transition of the calibration.

This query returns an integer value.

**Syntax**

:STAT:CAL:NTR

:STAT:CAL:NTR?

**:CALibration[:EVENT?]:PTRansition**  
**:CALibration[:EVENT?]:PTRansition?**

:CAL:PTR returns the state of the positive transition of the calibration.

This query returns an integer value.

**Syntax**

:STAT:CAL:PTR

:STAT:CAL:PTR?

**:COMMunicate:CONDition?**

This query returns an integer value.

**Syntax**

:STAT:COMM:COND?

**:COMMunicate:ENABLE**  
**:COMMunicate:ENABLE?**

This query returns an integer value.

**Syntax**

:STAT:COMM:ENAB

:STAT:COMM:ENAB?

**:COMMunicate[:EVENT?]:NTRansition**  
**:COMMunicate[:EVENT?]:NTRansition?**

This query returns an integer value.

**Syntax**

:STAT:COMM:NTR

:STAT:COMM:NTR?

**:COMMunicate[:EVENT?]:PTRansition**  
**:COMMunicate[:EVENT?]:PTRansition?**

This query returns an integer value.

**Syntax**

:STAT:COMM:PTR

:STAT:COMM:PTR?

**:HARD1:CONDition?**

Status reporting for hardware.

This query returns an integer value.

**Syntax**

:STATHARD1:COND?

**:HARD1:ENABLE**  
**:HARD1:ENABLE?**

This query returns an integer value.

**Syntax**

:STAT:HARD1:ENAB

:STAT:HARD1:ENAB?

**:HARD1[:EVENT?]:NTRansition**  
**:HARD1[:EVENT?]:NTRansition?**

This query returns an integer value.

**Syntax**

:STAT:HARD1:NTR

:STAT:HARD1:NTR?

**:HARD1[:EVENT?]:PTRansition**  
**:HARD1[:EVENT?]:PTRansition?**

This query returns an integer value.

**Syntax**

:STAT:HARD1:PTR

:STAT:HARD1:PTR?

**:HARD2:CONDition?**

This query returns an integer value.

**Syntax**

:STAT:HARD2:COND

:STAT:HARD2:COND?

**:HARD2:ENABLE**  
**:HARD2:ENABLE?**

This query returns an integer value.

**Syntax**

:STAT:HARD2:ENAB

:STAT:HARD2:ENAB?

**:HARD2[:EVENT?]:NTRansition**  
**:HARD2[:EVENT?]:NTRansition?**

This query returns an integer value.

**Syntax**

:STAT:HARD2:NTR

:STAT:HARD2:NTR?

**:HARD2[:EVENT?]:PTRansition**  
**:HARD2[:EVENT?]:PTRansition?**

This query returns an integer value.

**Syntax**

:STAT:HARD2:PTR

:STAT:HARD2:PTR?

### **:OPERation:CONDition?**

This query returns an integer value.

#### **Syntax**

:STAT:OPER:COND?

### **:OPERation:ENABLE :OPERation:ENABLE?**

This query returns an integer value.

#### **Syntax**

:STAT:OPER:ENAB

:STAT:OPER:ENAB?

### **:OPERation[:EVENT?]:NTRansition :OPERation[:EVENT?]:NTRansition?**

This query returns an integer value.

#### **Syntax**

:STAT:OPER:NTR

:STAT:OPER:NTR?

### **:OPERation[:EVENT?]:PTRansition :OPERation[:EVENT?]:PTRansition?**

This query returns an integer value.

#### **Syntax**

:STAT:OPER:PTR

:STAT:OPER:PTR?

### **:CALibrating:CONDition?**

This query returns an integer value.

#### **Syntax**

:STAT:CAL:COND?

**:CALibrating:ENABLE  
:CALibrating:ENABLE?**

This query returns an integer value.

**Syntax**

:STAT:CAL:ENAB

:STAT:CAL:ENAB

**:CALibrating[:EVENT?]:NTRansition  
:CALibrating[:EVENT?]:NTRansition?**

This query returns an integer value.

**Syntax**

:STAT:CAL:NTR

:STAT:CAL:NTR?

**:CALibrating[:EVENT?]:PTRansition  
:CALibrating[:EVENT?]:PTRansition?**

This query returns an integer value.

**Syntax**

:STAT:CAL:PTR

:STAT:CAL:PTR?

**:QUEStionable:CONDition?**

This query returns an integer value.

**Syntax**

:STAT:QUES:COND?

**:QUEStionable:ENABLE  
:QUEStionable:ENABLE?**

This query returns an integer value.

**Syntax**

:STAT:QUES:ENAB

:STAT:QUES:ENAB?



**:QUEStionable[:EVENT?]:NTRansition**  
**:QUEStionable[:EVENT?]:NTRansition?**

This query returns an integer value.

**Syntax**

:STAT:QUES:NTR

:STAT:QUES:NTR?

**:QUEStionable[:EVENT?]:PTRansition**  
**:QUEStionable[:EVENT?]:PTRansition?**

This query returns an integer value.

**Syntax**

:STAT:QUES:PTR

:STAT:QUES:PTR?

**:MEASuring:CONDition?**

This query returns an integer value.

**Syntax**

:STAT:MEAS:COND?

**:MEASuring:ENABle**  
**:MEASuring:ENABle?**

This query returns an integer value.

**Syntax**

:STAT:MEAS:ENAB

:STAT:MEAS:ENAB?

**:MEASuring[:EVENT?]:NTRansition**  
**:MEASuring[:EVENT?]:NTRansition?**

This query returns an integer value.

**Syntax**

:STAT:MEAS:NTR

:STAT:MEAS:NTR?

**:MEASuring[:EVENT?]:PTRansition**  
**:MEASuring[:EVENT?]:PTRansition?**

This query returns an integer value.

**Syntax**

:STAT:MEAS:PTR

:STAT:MEAS:PTR?

**:CDMA1:CONDition?**

This query returns an integer value.

**Syntax**

:STAT:CDMA1:COND?

**:CDMA1:ENABLE**  
**:CDMA1:ENABLE?**

This query returns an integer value.

**Syntax**

:STAT:CDMA1:ENAB

:STAT:CDMA1:ENAB?

**:CDMA1[:EVENT?]:NTRansition**  
**:CDMA1[:EVENT?]:NTRansition?**

This query returns an integer value.

**Syntax**

:STAT:CDMA1:NTR

:STAT:CDMA1:NTR?

**:CDMA1[:EVENT?]:PTRansition**  
**:CDMA1[:EVENT?]:PTRansition?**

This query returns an integer value.

**Syntax**

:STAT:CDMA1:PTR

:STAT:CDMA1:PTR?

### **:IBASic:CONDition?**

This query returns an integer value.

#### **Syntax**

:STAT:IBAS:COND?

### **:IBASic:ENABLE :IBASic:ENABLE?**

This query returns an integer value.

#### **Syntax**

:STAT:IBAS:ENAB

:STAT:IBAS:ENAB?

### **:IBASic[:EVENT?]:NTRansition :IBASic[:EVENT?]:NTRansition?**

This query returns an integer value.

#### **Syntax**

:STAT:IBAS:NTR

:STAT:IBAS:NTR?

### **:IBASic[:EVENT?]:PTRansition :IBASic[:EVENT?]:PTRansition?**

This query returns an integer value.

#### **Syntax**

:STAT:IBAS:PTR

:STAT:IBAS:PTR?

## **:SYSTem**

### **[:ERRor?]**

Returns an integer followed by quoted string.

#### **Syntax**

:SYST:ERR?

---

## **:TESTs Subsystem**

**:COMMENT1 | COMM1 <character data>**

**:COMMENT1? | COMM1?**

Allows a comment to be entered.

Allowed character set:

ABCDEFGHIJKLMNOPQRSTUVWXYZ\_012

3456789 abcdefghijklmnopqrstuvwxyz

!"#\$%&'()\*+,-./:;<=>?@[\\]^\_{|}~

### **Syntax**

:TEST:COMM1 'char text' ! 50 characters max

:TEST:COMM1?

**:COMMENT2 | COMM2**

**:COMMENT2? | COMM2?**

Allows a comment to be entered.

Valid character set:

ABCDEFGHIJKLMNOPQRSTUVWXYZ\_012

3456789 abcdefghijklmnopqrstuvwxyz

!"#\$%&'()\*+,-./:;<=>?@[\\]^\_{|}~

### **Syntax**

:TEST:COMM2 'char text' ! 50 characters max

:TEST:COMM2?

## **:CONFigure | CNFG**

**<integer\_value>,<character\_data>,<character\_data>,  
<integer\_value>,<character\_data>**

## **:CONFigure? | CNFG?**

:CONF allows you to configure external devices.

### **Syntax**

:TEST:CONF <integer>,'<string>','<string>,<integer>,'<string>'

:TEST:CONF? <integer> !returns unquoted string, 5 elements

! separated by commas

## **:EXECution:DESTination <string>**

## **:EXECution:DESTination?**

:EXEC:DEST sets/queries the destination of the output, either the display or the printer.

### **Syntax**

:TEST:EXEC:DEST 'CRT'

'Printer'

:TEST:EXEC:DEST?

## **:EXECution:FAILure <string>**

## **:EXECution:FAILure?**

:EXEC:FAIL defines what to do if a failure condition is encountered.

### **Syntax**

:TEST:EXEC:FAIL 'Continue'

'Stop'

:TEST:EXEC:FAIL

## **:EXECution:HEADING1 | HEAD1 <string>**

## **:EXECution:HEADING1? | HEAD1?**

Provides a heading.

### **Syntax**

:TEST:EXEC:HEAD1 '<string>'

:TEST:EXEC:HEAD1?

**:EXECution:HEADING2 | HEAD2 <string>**  
**:EXECution:HEADING2? | HEAD2?**

Provides a heading.

**Syntax**

:TEST:EXEC:HEAD2 '<string>'

:TEST:EXEC:HEAD2?

**:EXECution:RESults <string>**  
**:EXECution:RESults?**

:EXEC:RES sets/queries how to receive results from your program.

**Syntax**

:TEST:EXEC:RES 'All'

'Failures'

:TEST:EXEC:RES?

**:EXECution:RUN <string>**  
**:EXECution:RUN?**

:EXEC:RUN sets/queries how to run the program.

**Syntax**

:TEST:EXEC:RUN 'Continuous'

'Single Step'

:TEST:EXEC:RUN?

**:FREQuency <integer>,<real>,<character data>,<real>,<character data>,<string>,<string>**  
**:FREQuency?**

:FREQ consists of seven elements, made up of integers, real numbers, characters and strings.

**Syntax**

:TEST:FREQ <integer>,<real>,'<char>',<real>,'<char>', 'Yes', 'NO'

'NO', 'YES'

'Yes', 'YES'

'NO', 'NO'

:TEST:FREQ? !returns 7 elements in an unquoted string

## **:LIBRARY?**

Returns an unquoted string.

### **Syntax**

:TEST:LIBR?

## **:PARAMeter[:NUMBER] <integer>,<real>**

### **:PARAMeter[:NUMBER?]**

:PAR sets/queries numerical parameters.

### **Syntax**

:TEST:PAR <integer>,<real>

:TEST:PAR?

## **:PARAMeter:STRing <string>,<real>**

### **:PARAMeter:STRing?**

:PAR sets/queries string parameters.

### **Syntax**

:TEST:PAR:STR '<string>',<real>

:TEST:PAR:STR?

## **:PROCEDURE:AUTOstart <string>**

### **:PROCEDURE:AUTOstart?**

:PROC:AUT sets/queries the autostart feature of the Test Set.

### **Syntax**

:TEST:PROC:AUT 'ON'

'OFF'

:TEST:PROC:AUT?



**:PROCedure:LOCation <string>**  
**:PROCedure:LOCation?**

:PROC:LOC defines the location of the procedure.

**Syntax**

:TEST:PROC:LOC 'RAM'  
                  'ROM'  
                  Card'  
:TEST:PROC:LOC?

**:PROCedure:NAME <character data>**  
**:PROCedure:NAME?**

Name of the procedure.

Valid characters:

ABCDEFGHIJKLMNOPQRSTUVWXYZ\_0123456789

**Syntax**

:TEST:PROC:NAME '<char>' ! 10 characters max  
:TEST:PROC:NAME?

**:PROCedure:RUN**

Runs the named procedure.

**Syntax**

:TEST:PROC:RUN

**:PROCedure:RUNTest**

Runs the named procedure.

**Syntax**

:TEST:PROC:RUNT

**:SEQNumber[:NUMBER] <integer>,<string>**  
**:SEQNumber[:NUMBER?]**

Sequence Number.

Valid characters:

0123456789,YN

**Syntax**

:TEST:SEQN <integer>,'<string>' !max 249 characters

:TEST:SEQN?

**:SPEC[:NUMBER] <integer>,<real>,<real>,<string>**  
**:SPEC[:NUMBER?]**

Spec.

**Syntax**

:TEST:SPEC <integer>,<real>,<real>,'Upper'

'Lower'

'Both'

'None'

:TEST:SPEC?

**:SPEC:STRing <real>,<real>,<string>**  
**:SPEC:STRing?**

:SPEC:STR.

**Syntax**

:TEST:SPEC:STR <real>,<real>,'Upper'

'Lower'

'Both'

'None'

:TEST:SPEC:STR?

---

## **:TRIGger subsystem**

The Trigger subsystem contains commands for triggering measurements.

### **:ABORt**

:ABOR ends a measurement cycle in progress.

#### **Syntax**

:TRIG:ABOR

### **[:IMMediate]**

Triggers all active measurements in progress.

#### **Syntax**

:TRIG

### **:MODE:RETRigger :MODE:RETRigger?**

:MODE:RETR sets/queries the retriggering setting.

#### **Syntax**

:TRIG:MODE:RETR REPetitive

:TRIG:MODE:RETR SINGLE

:TRIG:MODE:RETR?

### **:MODE:SETTling :MODE:SETTling?**

:MODE:SETT sets/queries the transient settling function.

#### **Syntax**

:TRIG:MODE:MODE:SETT FAST

:TRIG:MODE:MODE:SETT FULL

:TRIG:MODE:MODE:SETT?



---

## **6 IEEE Common Commands**

This chapter describes IEEE 488.2 common commands.

---

## IEEE 488.2 Common Commands

The IEEE 488.2 Standard defines a set of common commands which provide for uniform communication between devices on the HP-IB. These commands are common to all instruments which comply with the IEEE 488.2 Standard. These commands control some of the basic instrument functions, such as instrument identification, instrument reset, and instrument status reporting.

The following common commands are implemented in the Test Set:

**Table 6-1**

<b>Mnemonic</b>	<b>Command Name</b>
*CLS	Clear Status Command
*ESE	Standard Event Status Enable Command
*ESE?	Standard Event Status Enable Query
*ESR?	Standard Event Status Register Query
*IDN?	Identification Query
*OPC	Operation Complete Command
*OPC?	Operation Complete Query
*OPT?	Option Identification Query
*PCB	Pass Control Back Command
*RCL	Recall Command
*RST	Reset Command
*SAV	Save Command
*SRE	Service Request Enable Command
*SRE?	Service Request Enable Query
*STB?	Read Status Byte Query
*TRG	Trigger Command
*TST?	Self-Test Query
*WAI	Wait-to-Continue Command

---

## Common Command Descriptions

### \*IDN? (Identification Query)

The \*IDN? query causes a device to send its identification information over the bus. The Test Set responds to the \*IDN? command by placing its identification information, in ASCII format, into the Output Queue. The response data is obtained by reading the Output Queue into a string variable of length 72. The response data is organized into four fields separated by commas. The field definitions are described in [table 6-2](#).

**Table 6-2** Device Identification

Field	Contents	Response from Test Set	Comments
1	Manufacturer	Hewlett-Packard	
2	Model	Depends upon Model of Test Set	
3	Serial Number	US12345678	ASCII character "0", decimal value 48, if not available
4	Firmware Revision Level	A.01.08	ASCII character "0", decimal value 48, if not available

---

**NOTE**

The Serial Number format can take one of two forms:

A = alpha character  
X = numeric character

The form returned will depend upon the manufacturing date of the Test Set being queried.

---

### Example BASIC program

#### Example

```
10 DIM A$(10)20 OUTPUT 714;"*IDN?"
30 ENTER 714;A$
40 PRINT A$
50 END
```

## \*OPT? (Option Identification Query)

The \*OPT? command tells the Test Set to identify any reportable device options installed in the unit. The Test Set responds to the \*OPT? command by placing information which describes any reportable installed options into the Output Queue. The data is in ASCII format. The response data is obtained by reading the Output Queue into a string variable of length 255. The response data is organized into fields separated by commas. Some fields, such as the Filter Option field, have more than one valid string (only one is returned).

If an option is not installed, an ASCII character 0 (zero) is placed in the output string for that option. If an option is standard, it is not reported (an ASCII character 0 (zero) is not placed in the output string for that option). The length of the returned string can vary depending upon the Test Set being queried, installed options and standard options.

### Example BASIC program

#### Example

```
10 DIM A$(255)20 OUTPUT 714;"*OPT?"
30 ENTER 714;A$
40 PRINT A$
50 END
```

## \*RST (Reset)

The \*RST command resets the Test Set. When the \*RST command is received the majority of fields in the Test Set are “restored” to a default value, some fields are “maintained” at their current state and some are “initialized” to a known state. Refer to ["Instrument Initialization" on page 326](#) for further information. Other operational characteristics are also affected by the \*RST command as follows:

- All pending operations are aborted.
- The Test Set's display screen is in the UNLOCKED state.
- Measurement triggering is set to TRIG:MODE:SETT FULL;RETR REP.
- Any previously received Operation Complete command (\*OPC) is cleared.
- Any previously received Operation Complete query command (\*OPC?) is cleared.
- The power-up self-test diagnostics are not performed.
- The contents of the SAVE/RECALL registers are not affected.
- Calibration data is not affected.



- The HP-IB interface is not reset (any pending Service Request is not cleared).
- All Enable registers are unaffected: Service Request, Standard Event, Communicate, Hardware #1, Hardware #2, Operation, Calibration, and Questionable Data/Signal.
- All Negative Transition Filter registers are unaffected: Communicate, Hardware #1, Hardware #2, Operational, Calibration, and Questionable Data/Signal.
- All Positive Transition Filter registers are unaffected: Communicate, Hardware #1, Hardware #2, Operational, Calibration, and Questionable Data/Signal.
- The contents of the RAM memory are unaffected.
- The contents of the Output Queue are unaffected.
- The contents of the Error Queue are unaffected.

### **\*TST? (Self-Test Query)**

The \*TST? self-test query causes the Test Set to execute a series of internal self-tests and place a numeric response into the Output Queue indicating whether or not the Test Set completed the self-test without any detected errors. The response data is obtained by reading the Output Queue into a numeric variable, real or integer. Upon successful completion of the self-test the Test Set settings are restored to their values prior to receipt of the \*TST? command.

---

**NOTE**

---

Refer to the Test Set's *Assembly Level Repair* manual for further information on Power-Up Self Test Failure.

### **Example BASIC program**

#### **Example**

```
10 INTEGER Slf_tst_respons20 OUTPUT 714;"*TST?"
30 ENTER 714;Slf_tst_respons
40 PRINT Slf_tst_respons$
50 END
```

### **Example response**

#### **Example**

512

## \*OPC (Operation Complete)

The \*OPC command allows for synchronization between the Test Set and an external controller. The \*OPC command causes the Test Set to set bit 0, Operation Complete, in the Standard Event Status Register to the TRUE, logic 1, state when the Test Set completes all pending operations. Detection of the Operation Complete message can be accomplished by continuous polling of the Standard Event Status Register using the \*ESR? common query command. However, using a service request eliminates the need to poll the Standard Event Status Register thereby freeing the controller to do other useful work.

---

### NOTE

The Test Set contains signal generation and signal measurement instrumentation. The instrument control processor is able to query the signal measurement instrumentation to determine if a measurement cycle has completed. However, the instrument control processor is not able to query the signal generation instrumentation to determine if the signal(s) have settled. In order to ensure that all signals have settled to proper values, the instrument control processor initiates a one-second delay upon receipt of the \*OPC, \*OPC? and \*WAI commands. In parallel with the one-second timer the instrument control processor commands all active measurements to tell it when the measurement(s) are done. When all active measurements are done and the one-second timer has elapsed, the \*OPC, \*OPC? and \*WAI commands are satisfied.

If the \*OPC, \*OPC? or \*WAI common commands are used to determine when a measurement has completed and the measurement is either in the OFF State or unavailable (four dashed lines on CRT display "- - - -"), the \*OPC, \*OPC? or \*WAI command will never complete.

---

## Example BASIC program - Service Request

### Example

```
10 OUTPUT 714;"*SRE 32"
20 !Enable SRQ on events in the Standard Event Status Register
30 OUTPUT 714;"*ESE 1"
40 !Enable Operation Complete bit in Standard Event Status Register
50 ON INTR 7,15 CALL Srvce_interupt!Set up interrupt
60 ENABLE INTR 7;2 !Enable SRQ interrupts
70 OUTPUT 714;"DISP RFG;RFG:OUTP 'Dupl';AMPL 0 dBm;FREQ 320 MHz;*OPC"
80 LOOP!Dummy loop to do nothing
90 DISP "I am in a dummy loop."
100 END LOOP
110 END
120 SUB Srvce_interupt
130 PRINT "All operations complete."
140 ! Note:
```

```
150 ! This interrupt service routine is not complete.
160 ! Refer to the "HP-IB Service Requests" on page 317 for
170 ! complete information.
180 SUBEND
```

The program enables bit 0 in the Standard Event Status Enable Register and also bit 5 in the Service Request Enable Register so that the Test Set will request service whenever the OPC event bit becomes true. After the service request is detected the program can take appropriate action. Refer to "HP-IB Service Requests" on page 317 for further information.

### **Example BASIC program - Polling the Standard Event Status Register**

#### **Example**

```
10 INTEGER Stdevnt_reg_val
20 OUTPUT 714;"DISP RFG;RFG:OUTP 'Dupl';AMPL 0 dBm;FREQ 320 MHz;*OPC"
30 LOOP
40 OUTPUT 714;"*ESR?" ! Poll the register
50 ENTER 714;Stdevnt_reg_val
60 EXIT IF BIT(Stdevnt_reg_val,0) ! Exit if Operation Complete bit set
70 END LOOP
80 PRINT "All operations complete."
90 END
```

## \*OPC? (Operation Complete Query)

The \*OPC? query allows for synchronization between the Test Set and an external controller by reading the Output Queue or by polling the Message Available (MAV) bit in the Status Byte Register. The \*OPC? query causes the Test Set to place an ASCII character, 1, into its Output Queue when the Test Set completes all pending operations. A consequence of this action is that the MAV bit in the Status Byte Register is set to the 1 state.

---

### NOTE

The Test Set contains signal generation and signal measurement instrumentation. The instrument control processor is able to query the signal measurement instrumentation to determine if a measurement cycle has completed. However, the instrument control processor is not able to query the signal generation instrumentation to determine if the signal(s) have settled. In order to ensure that all signals have settled to proper values, the instrument control processor initiates a one-second delay upon receipt of the \*OPC, \*OPC? and \*WAI commands. In parallel with the one-second timer the instrument control processor commands all active measurements to tell it when the measurement(s) are done. When all active measurements are done and the one-second timer has elapsed, the \*OPC, \*OPC? and \*WAI commands are satisfied.

If the \*OPC, \*OPC? or \*WAI common commands are used to determine when a measurement has completed and the measurement is either in the OFF State or unavailable (four dashed lines on CRT display "- - - -"), the \*OPC, \*OPC? or \*WAI command will never complete.

---

### Using the \*OPC? query by reading Output Queue

Bit 4 in the Service Request Enable Register is set to a value of zero (disabled). The \*OPC? query is sent to the Test Set at the end of a command message data stream. The application program then attempts to read the \*OPC? query response from the Test Set's Output Queue. The Test Set will not put a response to the \*OPC? query into the Output Queue until the commands have all finished.

---

### NOTE

Reading the response to the \*OPC? query has the penalty that both the HP-IB bus and the Active Controller handshake are in temporary holdoff state while the Active Controller waits to read the \*OPC? query response from the Test Set.

---

### Example BASIC program .

#### Example

```
10 INTEGER Output_que_val
20 OUTPUT 714;"*SRE 0"! Disable Service Requests
30 OUTPUT 714;"DISP RFG;RFG:OUTP 'Dupl';AMPL 0 dBm;FREQ 320 MHz;*OPC?"
40 ENTER 714;Output_que_val
```

```

50 !Program will wait here until all operations complete
60 PRINT "All operations complete."
70 END

```

### Using the \*OPC? query using the MAV bit in the Status Byte

Bit 4 in the Service Request Enable Register is set to a value of 1 (enabled). The \*OPC? query is sent to the Test Set at the end of a command message data stream. The Test Set will request service when the MAV bit in the Status Byte register is set to the TRUE, logic 1, state. After the service request is detected the application program can take appropriate action. Refer to the ["HP-IB Service Requests" on page 317](#) for further information.

### Example BASIC program .

#### Example

```

10 OUTPUT 714;"*SRE 16"
20 !Enable SRQ on data available in Output Queue (MAV bit)
30 ON INTR 7,15 CALL Srvic_e_interupt ! Set up interrupt
40 ENABLE INTR 7;2 ! Enable SRQ interrupts
50 OUTPUT 714;"DISP RFG;RFG:OUTP 'Dupl';AMPL 0 dBm;FREQ 320
MHz;*OPC?"
60 LOOP ! Dummy loop to do nothing
70 DISP "I am in a dummy loop."
80 END LOOP
90 END
100 SUB Srvic_e_interupt
110 ENTER 714;Output_que_val!Read the 1 returned by the *OPC?
120 ! query command
130 PRINT "All operations complete."
140 ! Note:
150 ! This interrupt service routine is not complete.
160 ! Refer to the "HP-IB Service Requests" on page 317 for
170 ! complete information.
180 SUBEND

```

## \*WAI (Wait To Complete)

The \*WAI command stops the Test Set from executing any further commands or queries until all commands or queries preceding the \*WAI command have completed.

### Example BASIC statement

#### Example

```
OUTPUT 714: "DISP RFG;RFG:OUTP 'Dupl';*WAI;AMPL 0 dBm"
```

---

#### NOTE

The Test Set contains signal generation and signal measurement instrumentation. The instrument control processor is able to query the signal measurement instrumentation to determine if a measurement cycle has completed. However, the instrument control processor is not able to query the signal generation instrumentation to determine if the signal(s) have settled. In order to ensure that all signals have settled to proper values, the instrument control processor initiates a one-second delay upon receipt of the \*OPC, \*OPC? and \*WAI commands. In parallel with the one-second timer the instrument control processor commands all active measurements to tell it when the measurement(s) are done. When all active measurements are done and the one-second timer has elapsed, the \*OPC, \*OPC? and \*WAI commands are satisfied.

If the \*OPC, \*OPC? or \*WAI common commands are used to determine when a measurement has completed and the measurement is either in the OFF State or unavailable (four dashed lines on CRT display "- - - -"), the \*OPC, \*OPC? or \*WAI command will never complete.

---

## \*CLS (Clear Status)

The \*CLS command clears the contents (sets all bits to zero) of all Event Registers summarized in the Status Byte. The \*CLS command also empties all queues (removes all current messages) which are summarized in the Status Byte, except the Output Queue. The Status Byte Register is cleared and any pending service request is cleared. The following Event Registers are affected:

- Hardware 1 Status Register
- Hardware 2 Status Register
- Questionable Data/Signal Register
- Standard Event Status Register
- Operational Status Register
- Calibration Status Register
- Communicate Status Register
- Call Processing Status Register

The Following message queues are affected:

- Error Message Queue

---

NOTE

The \*CLS command does not clear the contents of the Message Screen which is displayed on the CRT when the SHIFT RX keys are selected. This display is only cleared when the unit is powered on.

---

### **\*ESE (Standard Event Status Enable)**

The Test Set responds to the \*ESE command. See ["Status Reporting" on page 270](#) for a detailed explanation of the \*ESE command.

### **\*ESE? (Standard Event Status Enable Query)**

The Test Set responds to the \*ESE? command. See ["Status Reporting" on page 270](#) for a detailed explanation of the \*ESE? command.

### **\*ESR? (Standard Event Status Register Query)**

The Test Set responds to the \*ESR? command. See ["Status Reporting" on page 270](#) for a detailed explanation of the \*ESR? command.

### **\*SRE (Service Request Enable)**

The Test Set responds to the \*SRE command. ["HP-IB Service Requests" on page 317](#) for a detailed explanation of the \*SRE command.

### **\*SRE? (Service Request Enable Query)**

The Test Set responds to the \*SRE? command. ["HP-IB Service Requests" on page 317](#) for a detailed explanation of the \*SRE? command.

### **\*STB? (Status Byte Query)**

The Test Set responds to the \*STB? command. ["Status Reporting" on page 270](#) for a detailed explanation of the \*STB? command.

### **\*TRG (Trigger)**

The \*TRG command is equivalent to the IEEE 488.1 defined Group Execute Trigger (GET) message and has the same effect as a GET when received by the Test Set. The Test Set responds to the \*TRG command by triggering all currently active measurements.

### **\*PCB (Pass Control Back)**

The Test Set accepts the \*PCB command. ["Passing Control" on page 337](#) for a detailed explanation of the \*PCB command.

### **\*RCL (Recall Instrument State)**

The \*RCL command restores the state of the Test Set from a file previously stored in battery-backed internal memory, on a memory card, on a RAM disk, or on an external disk. The \*RCL command is followed by a decimal number in the range of 0 to 99 which indicates which Test Set SAVE/RECALL file to recall. The mass storage location for SAVE/RECALL files is selected using the SAVE/RECALL field on the I/O CONFIGURE screen.

The \*RCL command cannot be used to recall files with names which contain non-numeric characters or a decimal number greater than 99. To recall SAVE/RECALL files saved with names which contain non-numeric characters or a decimal number greater than 99, use the REG:RECall filename command.

### **\*SAV (Save Instrument State)**

The \*SAV command saves the present state of the Test Set into a file in battery-backed internal memory, on a memory card, on a RAM disk, or on an external disk. The \*SAV command is followed by a decimal number in the range of 0 to 99 which indicates the name of the stored SAVE/RECALL file. The mass storage location for SAVE/RECALL files is selected using the SAVE/RECALL field on the I/O CONFIGURE screen.

The \*SAV command cannot be used to save the present state of the Test Set to a file with a name which contains non-numeric characters or a decimal number greater than 99. To save the present state of the Test Set to a file with a name which contains non-numeric characters or a decimal number greater than 99, use the REG:SAVE filename command.



---

# 7

## Triggering Measurements

This chapter gives you information about triggering your measurements. This includes the following topics:

- ["Arming Hardware-Triggered Measurements" on page 262](#)
- ["Default Trigger Mode" on page 258](#)
- ["Local/Remote Triggering Changes" on page 259](#)
- ["Measurement Pacing" on page 262](#)
- ["Trigger Commands" on page 259](#)
- ["Trigger Event" on page 256](#)
- ["Trigger Mode and Measurement Speed" on page 260](#)
- ["Trigger Modes" on page 257](#)

## Triggering Measurements

The measurement cycle is started (triggered) by the occurrence of a trigger event. The reliability and accuracy of the measurement result, as well as the speed of the measurement cycle are influenced by the trigger mode in effect at the time the trigger event occurs. Some modes are faster than others; some modes provide settling for signals that may contain transients. The best triggering mode to use will depend upon the measurement requirements (repeatability, accuracy and speed).

### Trigger Event

The Test Set starts a measurement cycle when a valid Trigger Event is received. A Trigger Event is analogous to telling the Test Set to “start the measurement now.” There are three commands that can be used to issue a Trigger Event to the Test Set through HP-IB:

- A Group Execute Trigger Command (GET) as defined by IEEE 488.1-1987
- A Trigger Common Command (\*TRG) as defined by IEEE 488.2-1987
- A :TRIGger:IMMediate Test Set command.

All three commands are equivalent and have the same effect when received by the Test Set. The Test Set responds to the three commands by triggering *all* currently active measurements. A measurement is defined as *active* if

- it is on the currently displayed screen
- it is in the ON state

From a programming perspective this means that the screen which contains the measurement of interest must be made available using the DISPLAY command and that the measurement STATE must be ON.

## Trigger Modes

The Trigger Mode is defined by two parameters: retriggering and settling.

### Retriggering

Retriggering refers to what a measurement does once it has completed a measurement cycle. There are two options:

1. Single retriggering causes the measurement cycle to stop once a valid measurement result has been obtained. A valid trigger command must be received to start the measurement again. When a measurement cycle is completed, the values for all active measurements are held until another trigger command is received. This allows the control program to query a group of measurements that were triggered at the same time. This is the same functionality as the front-panel HOLD function.

When the trigger mode is set to single retriggering, consecutive queries of the same measurement (with no intervening trigger event) will return the same value. Measurements that rely on external signals or hardware-generated events (such as the DTMF Decoder) must be re-armed with a new trigger command before another measurement can be made.

2. Repetitive retriggering causes the measurement cycle to immediately start over once a valid measurement result has been obtained. No trigger event must be received to start the measurement again. Repetitive retriggering will cause measurements that rely on external signals or hardware generated events (such as the DTMF Decoder) to be re-armed upon completion of a measurement cycle (a valid measurement result has been obtained). When the trigger mode is set to repetitive retriggering, consecutive queries of the same measurement return new measured values.

---

#### NOTE

If a measurement cycle does not successfully obtain a valid measurement result, it will continue to try until it does or the measurement trigger is aborted. This is true for both retriggering modes.

---

## Settling

Settling refers to the amount of delay introduced to allow signal transients to propagate through the analysis chain and settle out.

There are two options:

1. Full settling introduces the appropriate delay for all signal transients which might have occurred at the front panel at just the same time as the trigger event, to pass through the analysis chain and settle out. Delays are also inserted to allow for internal hardware transients to settle.
2. Fast settling introduces no delay for internal or external signal transients to settle. The programmer must account for transient settling before issuing the Trigger Event.

---

### NOTE

There will still be delays introduced by the couplings between autotuning and autoranging. If the programmer wishes to remove these delays as well, all autoranging and autotuning functions must be turned OFF and the program must explicitly set the ranging amplifiers and the frequency tuning. Delays introduced by the measurement processes themselves cannot be eliminated.

---

### NOTE

#### Bus Lock Up:

If a measurement cycle does not successfully obtain a valid measurement result, it will continue to try until it does or until the measurement trigger is aborted. This is true for both retriggering modes. This has the consequence that both the HP-IB bus and the Active Controller handshake are in a temporary holdoff state while the Active Controller waits to read the measurement result from the Test Set.

The control program should include measurement time-out routines that CLEAR the bus and ABORT the trigger if a measurement does not complete within a specified amount of time. This provides a method of preventing the bus from remaining in the temporary holdoff state indefinitely.

---

## Default Trigger Mode

The Trigger mode is set to FULL SETTling and REPetitive RETRiggering whenever

- the Test Set is powered on
- the PRESET key is selected
- the Test Set is put into LOCAL mode
- the Test Set is reset using the \*RST command
- the Test Set is put in remote mode and no other trigger mode is set

## Local/Remote Triggering Changes

### Local To Remote Transitions

The Test Set switches from Local to Remote mode upon receipt of the Remote message (REN bus line true and Test Set is addressed to listen). No instrument settings are changed by the transition from Local to Remote mode, but triggering is set to the state it was last set to in Remote mode (if no previous setting, the default is FULL SETTling and REPetitive RETRiggering).

When the Test Set makes a transition from local to remote mode, all currently active measurements are flagged as invalid causing any currently available measurement results to become unavailable. If the HP-IB trigger mode is :RETR REP then a new measurement cycle is started and measurement results will be available for all active measurements when valid results have been obtained. If the HP-IB trigger mode is :RETR SING then a measurement cycle must be started by issuing a trigger event. Refer to "[Triggering Measurements](#)" on page 256 for more information.

### Remote To Local Transitions

The Test Set switches from Remote to Local mode upon receipt of the Local message (Go To Local bus message is sent and Test Set is addressed to listen) or receipt of the Clear Lockout/Set Local message (REN bus line false). No instrument settings are changed by the transition from Remote to Local mode, but triggering is reset to FULL SETTling and REPetitive RETRiggering.

## Trigger Commands

### :TRIGger:IMMediate

The :TRIGger:IMMediate command tells the Test Set to "start a measurement cycle now." The type of triggering used depends on the trigger mode settings. This command is equivalent to a Group Execute Trigger Command (GET) as defined by IEEE 488.1-1987 or a Trigger Common Command (\*TRG) as defined by IEEE 488.2-1987. The IMMediate statement is implied and is optional.

#### Syntax

```
:TRIGger:IMMediate
```

#### Example

```
OUTPUT 714;"TRIG:IMM"
```

or

```
OUTPUT 714;"TRIG"
```

## **:ABORt**

The **:ABORt** command tells the Test Set to stop a currently executing measurement cycle and get ready for a new HP-IB command. If for any reason a valid measurement cannot be made, this command allows the control program to terminate the requested measurement and regain control of the Test Set.

### **Syntax**

```
:TRIGger:ABORt
```

### **Example**

```
OUTPUT 714: "TRIG:ABOR"
```

## **:MODE**

The **:MODE** command is used to set the Trigger Mode for all active measurements. The trigger mode is defined by two parameters: retriggering and settling.

### **Retriggering Syntax**

```
:TRIGger:MODE:RETRigger REPetitive
```

```
:TRIGger:MODE:RETRigger SINGle
```

### **Retriggering Examples**

```
OUTPUT 714: "TRIG:MODE:RETR REP"
```

```
OUTPUT 714: "TRIG:MODE:RETR SING"
```

### **Settling Syntax**

```
:TRIGger:MODE:SETTling FAST
```

```
:TRIGger:MODE:SETTling FULL
```

### **Settling Examples**

```
OUTPUT 714: "TRIG:MODE:SETT FAST"
```

```
OUTPUT 714: "TRIG:MODE:SETT FULL"
```

## **Trigger Mode and Measurement Speed**

There are two generalized scenarios which can be described for HP-IB triggering control. The first is to have the Test Set return measurement results as fast as possible and assume that the control program will handle all transient settling and value tolerance activities. The second scenario is to have the Test Set return the most reliable, accurate, fully settled measurement results that it can, even if it takes some time to do this.

### **Trigger Mode Settings for Fastest Measurements**

Use the following Test Set configuration and trigger mode settings for the fastest possible measurement speed. See "[Increasing Measurement Throughput](#)" on page 264 for more information on improving measurement throughput.

1. Range hold all auto-ranging and auto-tuning functions and set ranges and frequency through HP-IB. This avoids autoranging/autotuning delays.
2. Use REPetitive RETRiggering. This avoids Trigger Event processing delays.
3. Use FAST SETTling. This avoids the signal transient settling delays.<sup>1</sup>
4. Turn off all measurements that are not required. This avoids any delays caused by contention for measurement resources within the Test Set.

### **Trigger Mode Settings for Most Reliable Measurements**

Use the following Test Set configuration and trigger mode settings to get the most accurate, most reliable, fully settled measurement results. See "[Increasing Measurement Throughput](#)" on page 264 for more information on improving measurement throughput.

1. Turn on all autoranging and autotuning functions. (This is the Test Set's default turn-on and PRESET state.)
2. Use SINGLE RETRiggering.
3. Use FULL SETTling.
4. Individually trigger each measurement.

---

1. Using FAST settling increases the possibility that transient signal conditions which occur during the measurement cycle will be included in the measurement result.

## Measurement Pacing

Measurement pacing can be accomplished by using the IEEE 488.2-1987 Common Commands \*OPC, \*OPC?, and \*WAI. These commands are implemented within the Test Set using the criteria that an operation has not completed until

- all active measurements have obtained at least one valid measurement result
- all signals generated by the Test Set are within specifications.

Refer to the "[Common Command Descriptions](#)" on page 245 and the IEEE 488.2-1987 Standard for more information on using these commands.

## Arming Hardware-Triggered Measurements

Some measurements, such as the Tone Sequence Decoder, require an external signal to trigger the measurement. These measurements require that the measurement be "armed" in order for it to be triggered by the external signal. The :TRIGger:IMMediate command is used to arm these types of measurements within the Test Set.

When the trigger mode is set to RETRigger SINGle, the measurement must be re-armed after each measurement cycle.

When the trigger mode is set to RETRigger REPetitive, the measurement is continually re-armed after each measurement cycle.

---

### NOTE

#### Bus Lock Up:

If the required triggering signal is not received, or if the signal level is incorrect, the measurement will not trigger and the measurement cycle will not complete. If a measurement cycle does not successfully obtain a valid measurement result, it will continue to try until it does (an external trigger is detected) or until the measurement trigger is aborted. This is true for both retriggering modes. This has the consequence that both the HP-IB bus and the Active Controller handshake are in a temporary holdoff state while the Active Controller waits to read the measurement result from the Test Set.

The control program should include measurement time-out routines that CLEAR the bus and ABORT the trigger if a measurement does not complete within a specified amount of time. This provides a method of preventing the bus from remaining in the temporary holdoff state indefinitely.

---



---

## **8** **Advanced Operations**

## Increasing Measurement Throughput

Measurement throughput is defined as the number of measurements made per unit of time. When operating the Test Set in the Internal or External Automatic Control Mode, measurement throughput is influenced by measurement speed, measurement setup time, and execution speed of the control program. Each of these factors is, in turn, influenced by several parameters. The following sections discuss the parameters and their effect on measurement throughput.

### Optimizing Measurement Speed

Measurement speed is defined as the time required to complete one measurement cycle after receipt of a valid trigger event. Measurement speed is influenced by the following four parameters.

#### 1. Trigger Mode

The Trigger Mode affects the time-to-first-reading and the length of the measurement cycle and is defined by two parameters: retriggering and settling. Retriggering refers to what a measurement does once it has completed a measurement cycle. Settling refers to the amount of delay introduced to allow signal transients to propagate through the analysis chain and settle out. Refer to "[Triggering Measurements](#)" on page 255 for information on Trigger Mode and its impact on measurement speed.

#### 2. Autoranging/Autotuning

The autoranging and autotuning functions continuously calculate and adjust gain and frequency tuning settings to provide the optimum instrument setup for each measurement. This results in greater measurement accuracy but increases measurement cycle time. The autoranging and autotuning functions can be turned off to decrease the measurement cycle time.

Time-to-first-reading after making new settings is usually much slower than the repetitive reading rate once the first reading has been returned. The main contributor to first-reading measurement time is hardware autoranging. Hardware autoranging time can be eliminated by first establishing the expected AF and RF signal levels into the Test Set. With these signal levels present, the Test Set will autorange, allowing the operator to determine the attenuation and gain settings of the RF input attenuator as displayed in the RF ANALYZER screen, and to determine the various IF and audio gains as displayed in the AF ANALYZER screen. The attenuation and gain settings determined in manual mode should be recorded for use in writing the program.

In the control program, select Gain Control, Hold (default is Auto), and make the settings recorded in manual mode. When the control program runs, the signal levels into the Test Set need to remain relatively constant since autoranging has been disabled.

If the automatic functions are turned off, the control program must set the gain stages and frequency tuning before triggering a measurement. The automatic functions can be turned off as follows:

Disable RF autotuning by setting the `Tune Mode` field to `Manual` using the following command:

**Example**

```
:RFAN:TMOD 'MANUAL'
```

Disable RF autoranging by setting the `Input Atten` field to `Hold` using the following command:

**Example**

```
:RFAN:ATT:MODE 'HOLD'
```

Disable AF autoranging by setting the `Gain Cntl` field to `Hold` using the following command:

**Example**

```
:AFAN:RANG 'HOLD'
```

### 3. Frequency Counter Gate Time

The frequency counter's gate time specifies how long the RF or AF frequency counter samples the signal before displaying the measured result. Short gate times measure instantaneous frequency and long gate times measure average frequency. The longer the gate time, the longer the measurement cycle. The proper gate time is determined by the measurement requirements. Use the following commands to set gate times:

For AF frequency measurements, set the AF Analyzer's gate time with the `AF Cnt Gate` field, using the following command:

**Example**

```
:AFAN:GTIM <value> MS
```

For RF frequency measurements, set the RF Analyzer's gate time with the `RF Cnt Gate` field, using the following command:

**Example**

```
:RFAN:GTIM <value> MS
```

#### **4. Number of Active Measurements**

The Test Set is capable of making many measurements simultaneously. Measurements are either in the active state (ON) or in the inactive state (OFF). When the Test Set receives a trigger event, all active measurements are triggered. A measurement cycle is complete when all active measurements have obtained a valid measurement result. To decrease the measurement cycle time, all unused measurements should be set to the inactive state (turned OFF). Turning OFF unused measurements will have the greatest impact on reading repetition rate. Use the STATE command to turn OFF all unneeded measurements on the displayed screen.

### **Optimizing Measurement Setup Time**

Measurement setup time is defined as the time required to configure an individual instrument within the Test Set to make a measurement.

In general there are two methodologies which can be used to setup individual instruments in the Test Set:

1. Set up every field every time a measurement is made.
2. Define a base instrument state and then modify it as needed for each measurement (always returning to the base state after finishing the measurement).

Defining a base instrument state requires fewer HP-IB transactions to set up an instrument (in the majority of cases) which in turn reduces measurement setup time.

### **Optimizing the Execution Speed of the Control Program**

Execution speed of the control program is defined as the time required to execute a given number of program lines.

Each time the HP-IB is accessed, a given amount of time is required to configure the devices on the bus for data transfer. Every time a BASIC or IBASIC OUTPUT or ENTER statement is executed this bus configuration time is incurred. The total amount of bus configuration time expended for a given number of program lines can be minimized by reducing the number of OUTPUT and ENTER statements used in the control program. This is accomplished by combining several commands into one HP-IB transaction. Execution speed of the control program is influenced by the use of compound commands and screen display time as described in the following paragraphs.

## Compound Commands for Combining OUTPUT Statements

To reduce the number of OUTPUT statements used to make the desired settings within one screen, string together multiple settings within one OUTPUT statement. This is accomplished using the ; (semicolon) separator and the :: (semicolon colon) separator.

**The ; (semicolon) Separator.** The ; (semicolon) separator tells the Test Set's HP-IB command parser to back up *one* level of command hierarchy and accept the next command at the same level as the previous command. The following examples illustrate proper use of the semicolon separator:

### Example #1

```
OUTPUT 714;"RFG:AMPL -66 DBM;FREQ 500 MHZ;AMPL:STAT ON"
!This OUTPUT statement sets the RF generator's
amplitude, frequency, and output state.
```

### Example #2

```
OUTPUT 714;"RFG:MOD:EXT:DEST 'FM (/Vpk)';FM 12.5 KHZ;FM:STAT ON"
!This OUTPUT statement configures the RF generator to
accept external modulation from the rear-panel input,
sets the amount of deviation, and turns FM on.
```

### Example #3

```
OUTPUT 714;"ENC:AMPS:SAT:FM 2.35 KHZ;FREQ 5.970 KHZ"
!This OUTPUT statement sets the AMPS SAT tone's
frequency and deviation.
```

The semicolon separator tells the Test Set's HP-IB command parser to back up *only one* level of command hierarchy. The following OUTPUT statement illustrates *improper* use of the semicolon separator.

### Example

```
OUTPUT 714;"RFG:MOD:EXT:DEST 'FM (/Vpk)';AOUT 'DC'"
```

Trying to execute this OUTPUT statement would cause HP-IB Error:-113 Undefined header. This is because the AOUT command is two levels higher than the DEST 'FM (/Vpk)' command. See [":RFGenerator subsystem" on page 219](#) for the command hierarchy.

**The ;: (semicolon-colon) Separator.** The ;: (semicolon-colon) separator tells the Test Set's HP-IB command parser that the next command is at the top level of the command hierarchy. This allows commands from different instruments to be output on one command line. The following example illustrates proper use of the semicolon-colon separator:

#### Example

```
OUTPUT 714;"RFAN:FREQ 850 MHZ;;AFAN:INP 'FM DEMOD'"
```

This OUTPUT statement sets the RF Analyzer's tune frequency to 850 MHz, and then sets the AF Analyzer's input to FM Demod.

### Compound Commands for Combining ENTER Statements

To reduce the number of ENTER statements used to read measured values within one screen, string together multiple measure commands within one OUTPUT statement followed by an ENTER statement with the appropriate number of variables to hold the measured values. The following example illustrates this technique.

#### Example

```
OUTPUT 714;"MEAS:RFR:POW?;FREQ:ABS?"  
ENTER 714;Power,Freq_abs
```

This OUTPUT statement requests an RF power and an absolute RF frequency measurement. The ENTER statement then reads both values into program variables.

### Screen Display Time

Each time the screen being displayed on the Test Set (active screen) is changed, it takes approximately 250 ms to access and draw the new screen. Additionally, each time a field on the active screen is changed it takes a finite amount of time to update the field. This update time is depends on the length and type of field.

When the Test Set is operated in the External Automatic Control mode it is possible to program it to only display a special "remote screen." When the Test Set is in this mode, only the remote screen is displayed, regardless of the screen changes requested by the control program running on the external controller. The remote screen mode has no affect on the operation of the Test Set; all measurement modes, commands, and so forth, still operate normally. The only effect is that the active screen is never changed. This saves approximately 250 ms for every :DISPlay command in the control program and some finite amount of time for every field update.

When the remote screen is being displayed, the Test Set's display is in the "locked" state. When the control program on the external controller unlocks the display, the screen is returned to the last remote screen requested. The \*RST Common Command will place the Test Set's

display screen in the “unlocked” state. Pressing the LOCAL key causes a locked Test Set display session to end and causes the Test Set’s display screen to return to the unlocked state.

**To Lock the Test Set’s Display Screen.** Use the :SPECial:DISPlay 'LOCKED'|'UNLOCKED' commands to lock and unlock the Test Set’s display screen. 'LOCKED' causes the display to go to the remote screen and ignore screen changes and other display updates thus saving time. 'UNLOCKED' causes the display to return to the last screen requested by the remoter user. Further screen changes and display updates take place as normal.

#### Syntax

```
:SPECiaL:DISPlay 'LOCKED'  
:SPECiaL:DISPlay 'UNLOCKED'
```

#### Example

```
OUTPUT 714;"SPEC:DISP 'LOCKED'
```

This locks the Test Set’s display screen.

**To Query the Lock/Unlock State of the Test Set’s Display Screen.** Use the :SPECial:DISPlay? commands to query the lock/unlock state of the Test Set display screen.

#### Syntax

```
:SPECiaL:DISPlay?
```

#### Example

```
OUTPUT 714;"SPEC:DISP?"  
ENTER 714;Lock_unlock$
```

This queries the lock/unlock state of the Test Set’s display screen.

---

#### NOTE

Locking and unlocking the Test Set’s display screen is an External Automatic Control mode, HP-IB only function and cannot be done through the Test Set’s front panel.

SPECial:DISPlay 'LOCKED'|'UNLOCKED' must not be invoked from the Test Set’s built-in IBASIC Controller. Executing these commands from an IBASIC program can produce unexpected results and is not supported.

---

## Status Reporting

This section describes the status reporting structure used in the Test Set. The structure is based on the IEEE 488.1-1987 and 488.2-1987 Standards and the Standard Commands for Programmable Instruments (SCPI) Version 1994.0.

### Status Reporting Structure Overview

[Figure 8-1 on page 271](#) shows an overview of the status reporting structure used in the Test Set. The status reporting structure is used to communicate the Test Set's current status information to the application program. The term *status information* encompasses a variety of conditions which can occur in a Test Set, such as, has a measurement been completed, has an internal hardware failure occurred, has a command error occurred, has data available, and so forth. Many such conditions can exist in the Test Set. Like conditions are grouped together and maintained in Status Register Groups. Information in each register group is summarized into a Summary Message Bit. Summary Message Bits always track the current status of the associated register group. All of the Summary Message Bits are, in turn, summarized into the Status Byte Register.

Therefore, by monitoring the bits in the Status Byte Register the application program can determine if a condition has occurred which needs attention, which register to interrogate to determine what condition(s) have occurred, and what action to take in response to the condition.

---

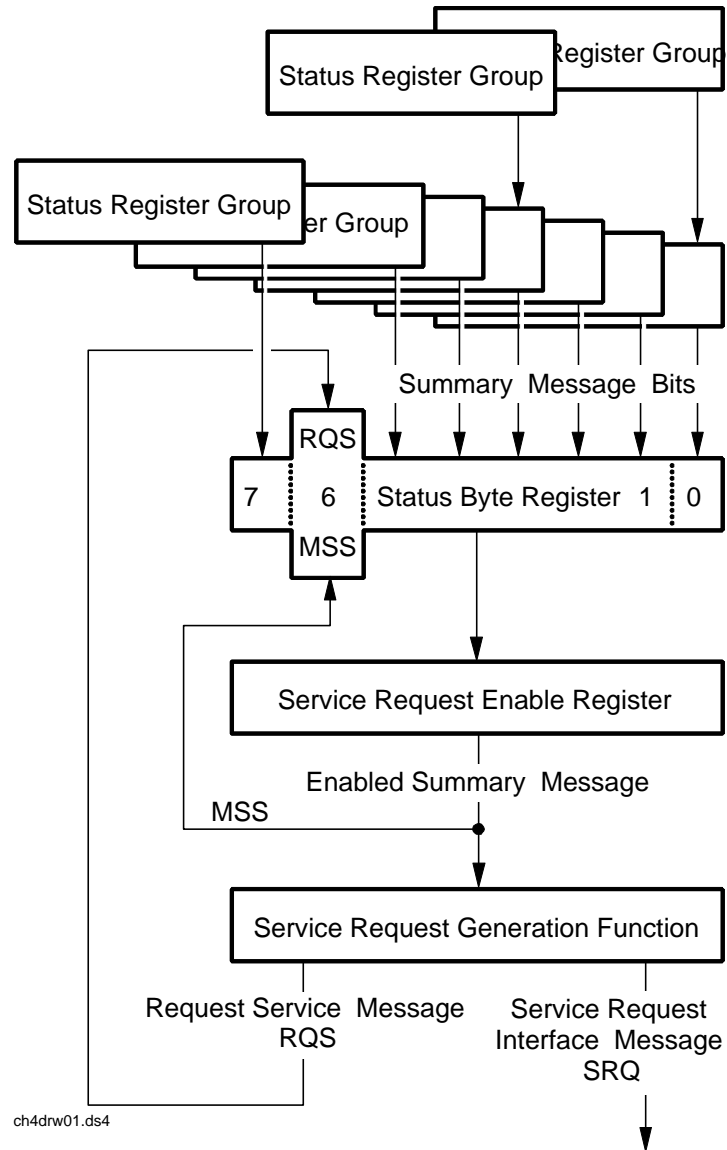
NOTE

A Status Register Group Summary Message Bit may be summarized indirectly to the Status Byte Register through a Status Register Group which is summarized directly into the Status Byte Register.

Bits in the Status Byte Register can also be used to initiate a Service Request message (SRQ) by enabling the associated bit in the Service Request Enable Register. When an enabled condition exists, the Test Set sends the Service Request message (SRQ) on the HP-IB bus and reports that it has requested service by setting the Request Service (RQS) bit in the Status Byte Register to the TRUE, logic 1, state. The Service Request message (SRQ) capability of the HP-IB bus is used to automatically signal the Active Controller that the Test Set needs attention. The application program can then interrogate the Test Set and determine what caused it to request service. Refer to ["HP-IB Service Requests" on page 317](#) for information on setting up, enabling and servicing SRQ generated interrupts.



**Figure 8-1**      **Status Reporting Structure Overview**



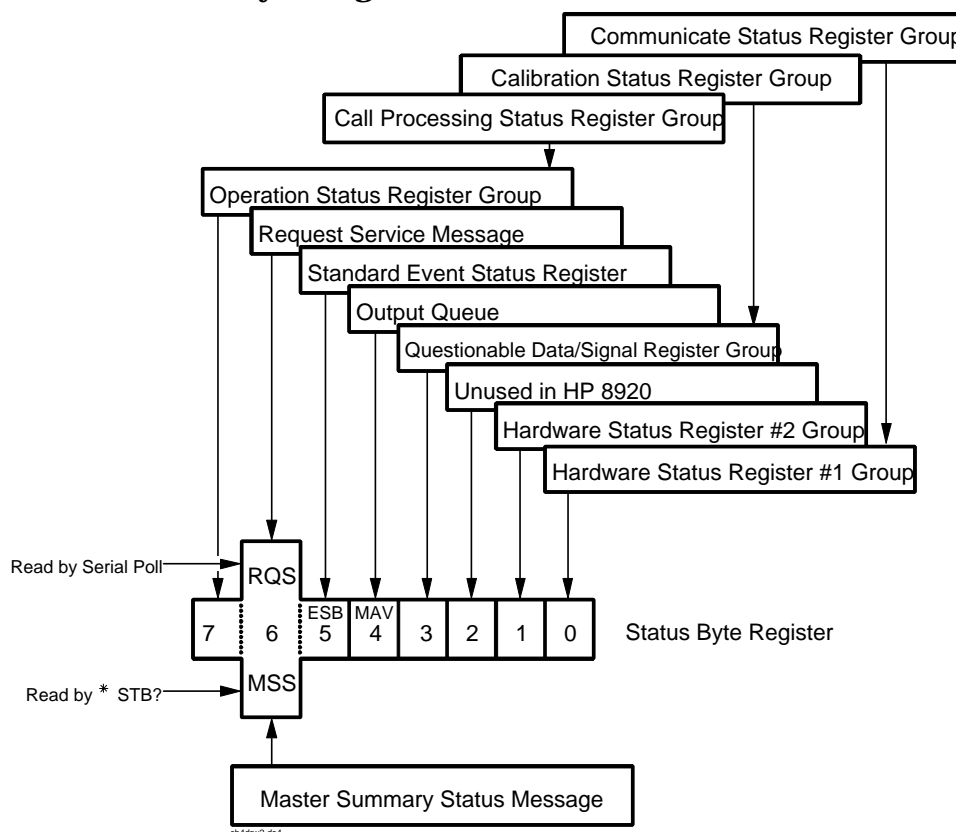
HP 8920 Status Reporting Structure

## Status Byte Register

The Status Byte Register is an 8-bit register that is used to summarize the Summary Messages from all the register groups in the Test Set, and the Request Service (RQS) or Master Summary Status (MSS) messages. The contents of the Status Byte Register, referred to as the *status byte*, can be read by the Active Controller to determine the condition of each of the register groups. The Summary Message from each register group is assigned to a specific bit position in the Status Byte Register as shown in [figure 8-2](#). If the Summary Message from a particular register group is TRUE, logic 1, its assigned bit in the Status Byte Register will also be TRUE. If the Summary Bit from a particular register group is FALSE, logic 0, its assigned bit in the Status Byte Register will also be FALSE.

**NOTE** A Status Register Group Summary Message Bit may be summarized indirectly to the Status Byte Register through a Status Register Group which is summarized directly into the Status Byte Register.

**Figure 8-2** Status Byte Register



[Table 8-1](#) details the Status Byte Register bit assignments and their associated meaning.

**Table 8-1 Status Byte Register Bit Assignments**

Bit Number	Binary Weighting	Condition	Comment
7	128	Operation Status Register Group Summary Message	1= one or more of the enabled events have occurred since the last reading or clearing of the Event Register
6	64	Request Service (RQS) when read by serial poll OR Master Summary Status message when read by *STB? command	1= Test Set has requested service OR 1= one or more of the enabled service request conditions is true
5	32	Standard Event Status Bit (ESB) Summary Message	1= one or more of the enabled events have occurred since the last reading or clearing of the Event Register
4	16	Output Queue Message Available (MAV) Summary Message	1= information is available in the Output Queue
3	8	Questionable Data/Signal Register Group Summary Message	1= one or more of the enabled events have occurred since the last reading or clearing of the Event Register
2	4	Unused in Test Set	
1	2	Hardware #2 Status Register Group Summary Message	1 = one or more of the enabled events have occurred since the last reading or clearing of the Event Register
0	1	Hardware #1 Status Register Group Summary Message	1 = one or more of the enabled events have occurred since the last reading or clearing of the Event Register

The Status Byte Register is unique in that bit 6 can take on two different meanings. The contents of the Status Byte Register can be read two ways: by using a serial poll, or by using the \*STB? Common Command. Both methods return the status byte message, bits 0-5, and bit 7, as defined in [table 8-1](#). The value sent for bit 6 is, however, dependent upon the method used.

### Reading with a Serial Poll

The contents of the Status Byte Register can be read by a serial poll from the Active Controller in response to some device on the bus sending the Service Request (SRQ) message. When read with a serial poll, bit 6 in the Status Byte Register represents the Request Service (RQS) condition. Bit 6 is TRUE, logic 1, if the Test Set is sending the Service Request (SRQ) message and FALSE, logic 0, if it is not. Bits 0-5 and bit 7 are defined as shown in [table 8-1 on page 273](#). When read by a serial poll the RQS bit is cleared (set to 0) so that the RQS message will be FALSE if the Test Set is polled again before a new reason for requesting service has occurred. Bits 0-5 and bit 7 are unaffected by a serial poll.

### Reading with the \*STB? Common Command

The contents of the Status Byte Register can be read by the application program using the \*STB? Common Command. When read with the \*STB? Common Command, bit 6 represents the Master Summary Status (MSS) message. The MSS message is the inclusive OR of the bitwise combination (excluding bit 6) of the Status Byte Register and the Service Request Enable Register. For a discussion of Summary Messages, see "[Status Register Structure Overview](#)" on [page 276](#). Bit 6 is TRUE, logic 1, if the Test Set has at least one reason for requesting service and FALSE, logic 0, if it does not. Bits 0-5 and bit 7 are defined as shown in [table 8-1 on page 273](#). When read by the \*STB? Common Command, bits 0-5, bit 6, and bit 7 are unaffected.

The \*STB? Status Byte Query allows the programmer to determine the current contents (bit pattern) of the Status Byte Register and the Master Summary Status (MSS) message as a single element. The Test Set responds to the \*STB? query by placing the binary-weighted decimal value of the Status Byte Register and the MSS message into the Output Queue. The response represents the sum of the binary-weighted values of the Status Byte Register's bits 0-5 and 7 (weights 1,2,4,8,16,32 and 128 respectively) and the MSS summary message (weight 64). Thus, the response to \*STB?, when considered as a binary value, is identical to the response to a serial poll except that the MSS message of 1 indicates that the Test Set has at least one reason for requesting service (Refer to the IEEE 488.2-1987 Standard for a complete description of the MSS message). The decimal value of the bit pattern will be a positive integer in the range of 0 to 255. The response data is obtained by reading the Output Queue into a numeric variable, integer or real.

## Example BASIC program to read Status Byte with \*STB command

### Example

```
10 INTEGER Stat_byte_reg,Stat_byte,Mstr_sum_msg
20 OUTPUT 714;"*STB?"
30 ENTER 714;Stat_byte_reg
40 Stat_byte=BINAND(Stat_byte_reg,191) !mask out the MSS bit
50 PRINT Stat_byte
60 Mstr_sum_msg=BINAND(Stat_byte_reg,64) !mask out the Stat Byte
70 PRINT Mstr_sum_msg
80 END
```

### Example

```
32
0
```

## Writing the Status Byte Register

The Status Byte Register is a read-only register and is altered only when the state of the Summary Messages from the overlaying data structures are altered.

## Clearing the Status Byte Register

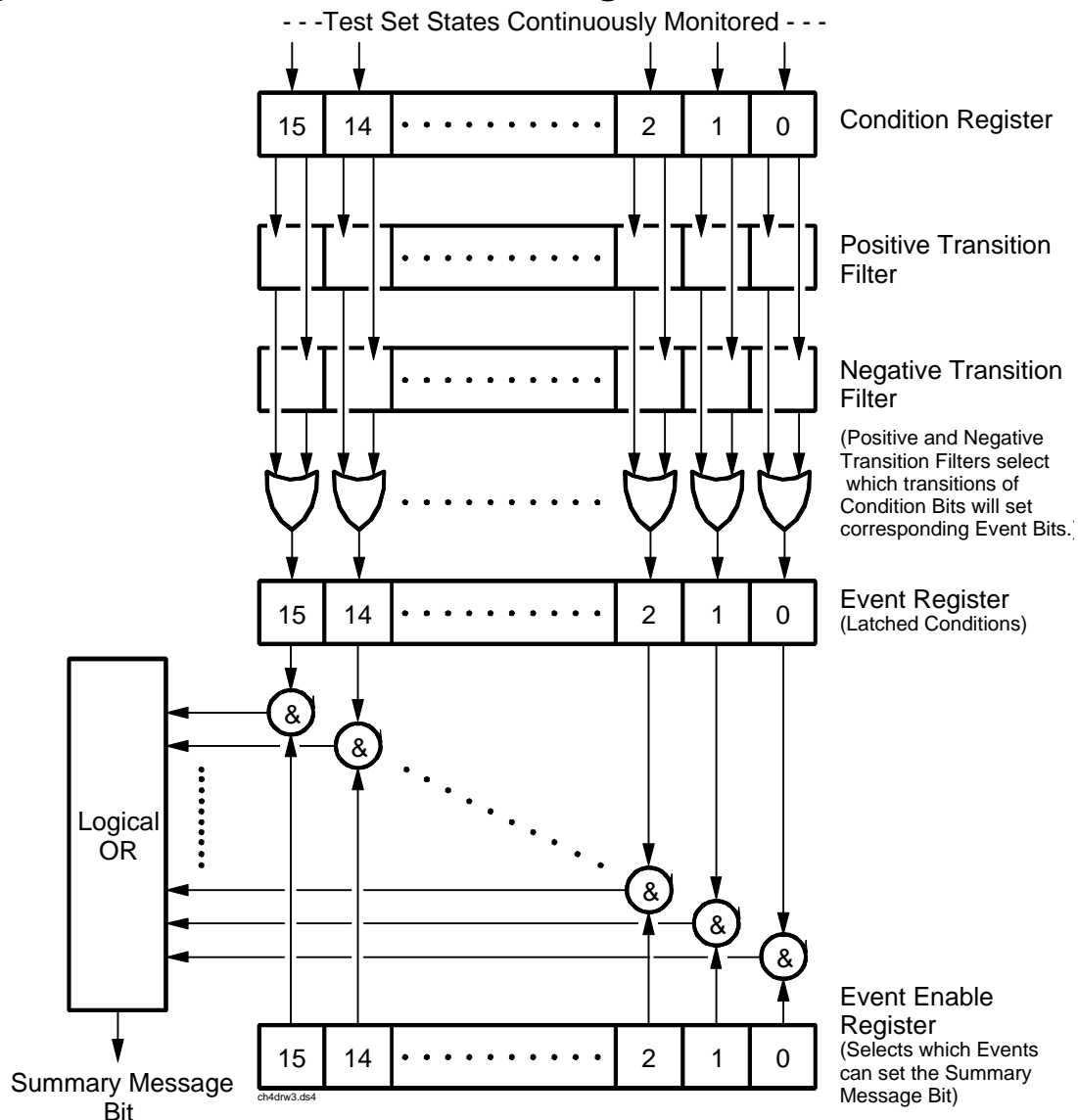
The \*CLS Common Command clears all Event Registers and Queues so that their corresponding Summary Messages are cleared. The Output Queue and its MAV Summary Message are an exception and are unaffected by the \*CLS Common Command.

## Status Register Structure Overview

The structure of the register groups used in the Test Set is based upon the status data structures outlined in the IEEE 488 and SCPI 1994.0 Standards. There are two types of status data structures used in the Test Set: status registers and status queues. The general models, components, and operation of each type of status data structure are explained in the following sections.

**Figure 8-3**

### Status Data Structure - Register Model



### Status Register Model

This section explains how the status registers are structured in the Test Set. The generalized status register model shown in [figure 8-2 on page 287](#) is the basis upon which all the status registers in the Test Set are built.

The model consists of a Condition Register, Transition Filters, an Event Register, an Enable Register, and a Summary Message. A set of these registers is called a Status Register Group.

**Condition Register.** A condition is a Test Set state that is either TRUE or FALSE (an HP-IB command error has occurred or an HP-IB command error has not occurred). Each bit in a Condition Register is assigned to a particular Test Set state. A Condition Register continuously monitors the hardware and firmware states assigned to it. There is no latching or buffering of any bits in a Condition Register; it is updated in real time. Condition Registers are read-only. Condition Registers in the Test Set are 16 bits long and may contain unused bits. All unused bits return a zero value when read.

**Transition Filters.** For each bit in the Condition Register, the Transition Filters determine which of two bit-state transitions will set the corresponding bit in the Event Register. Transition Filters may be set to pass positive transitions (PTR), negative transitions (NTR) or either (PTR or NTR). A positive transition means a condition bit changed from 0 to 1. A negative transition means a condition bit changed from 1 to 0. . In the Test Set, the Transition Filters are implemented as two registers: a 16-bit positive transition (PTR) register and a 16-bit negative transition (NTR) register. A positive transition of a bit in the Condition register will be latched in the Event Register if the corresponding bit in the positive transition filter is set to 1. A positive transition of a bit in the Condition register will *not* be latched in the Event Register if the corresponding bit in the positive transition filter is set to 0. A negative transition of a bit in the Condition register will be latched in the Event Register if the corresponding bit in the negative transition filter is set to 1. A negative transition of a bit in the Condition register will *not* be latched in the Event Register if the corresponding bit in the negative transition filter is set to 0. Either transition (PTR or NTR) of a bit in the Condition Register will be latched in the Event Register if the corresponding bit in both transition filters is set to 1. No transitions (PTR or NTR) of a bit in the Condition Register will be latched in the Event Register if the corresponding bit in both transition filters is set to 0.

Transition Filters are read-write. Transition Filters are unaffected by a \*CLS (clear status) command or queries. The Transitions Filters are set to pass positive transitions (PTR) at power on and after receiving the \*RST (reset) command (all 16 bits of the PTR register set to 1 and all 16 bits of the NTR register set to 0).

**Event Register.** . The Event Register captures bit-state transitions in the Condition Register as defined by the Transition Filters. Each bit in the Event Register corresponds to a bit in the Condition Register, or if there is no Condition Register/Transition Filter combination, each bit corresponds to a specific condition in the Test Set. Bits in the Event Register are latched, and, once set, they remain set until cleared by a query of the Event Register or a \*CLS (clear status) command. This guarantees that the application can't miss a bit-state transition in the Condition Register. There is no buffering; so while an event bit is set, subsequent transitions in the Condition Register corresponding to that bit are ignored. Event Registers are read-only. Event Registers in the Test Set are either 8 or 16 bits long and may contain unused bits. All unused bits return a zero value when read.

**Event Enable Register.** . The Event Enable Register defines which bits in the Event Register will be used to generate the Summary Message. Each bit in the Enable Register has a corresponding bit in the Event Register. The Test Set logically ANDs corresponding bits in the Event and Enable registers and then performs an inclusive OR on all the resulting bits to generate the Summary Message. By using the enable bits the application program can direct the Test Set to set the Summary Message to the 1 or TRUE state for a single event or an inclusive OR of any group of events. Enable Registers are read-write. Enable Registers in the Test Set are either 8 or 16 bits long and may contain unused bits which correspond to unused bits in the associated Event Register. All unused bits return a zero value when read and are ignored when written to. Enable Registers are unaffected by a \*CLS (clear status) command or queries.

**Summary Message Bit.** The Summary Message is a single-bit message which indicates whether or not one or more of the enabled events have occurred since the last reading or clearing of the Event Register. The Test Set logically ANDs corresponding bits in the Event and Enable registers and then performs an inclusive OR on all the resulting bits to generate the Summary Message. By use of the enable bits, the application program can direct the Test Set to set the Summary Message to the 1, or TRUE, state for a single event or an inclusive OR of any group of events. The Summary Message is TRUE when an enabled event in the Event Register is set TRUE. Conversely, the Summary Message is FALSE when no enabled events are TRUE. Summary Messages are always seen as bits in another register.



## Status Reporting Structure Operation.

In general the status reporting structure described on the previous pages is used as follows:

- Determine which conditions, as defined by their bit positions in the Condition Register, should cause the Summary Message to be set TRUE if they occur.

For example, Condition Register Bit 3 = Overpower Protection Tripped

- Determine the polarity of the bit-state transition which will indicate the condition has occurred.

For example,

- logic 0 = Overpower Protection not tripped
- logic 1 = Overpower Protection tripped
- occurrence indicated by a 0 to 1 transition
- use positive transition (PTR) filter for bit 3
- Set the Transition Filters to the correct polarity to pass the bit-state transition to the Event Register.

For example,

1. Set Positive Transition Filter bit 3 to 1, all other bits to 0.
  2. Set Negative Transition Filter bit 3 to 0, all other bits to 0.
- Set the correct bits in the Enable Register to generate the Summary Message if the condition has been latched into the Event Register.

For example,

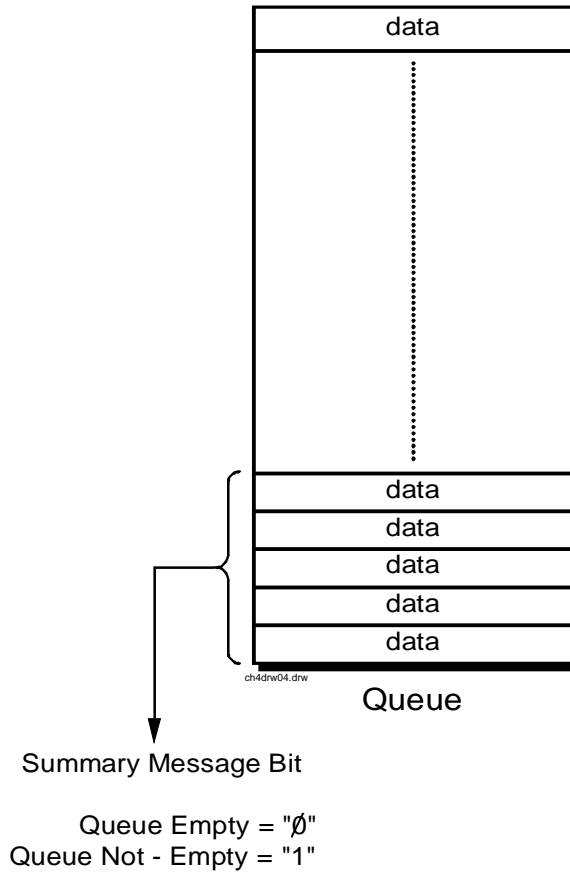
Set bit 3 of the Enable Register to a logic 1, all other bits to 0.

- Repeat these steps for any register containing the Summary Message bit.

## Status Queue Model

This section explains how status queues are structured in the Test Set. The generalized status queue model shown in [figure 8-4](#) is the basis upon which all the status queues in the Test Set are built. A queue is a data structure containing a sequential list of information. The queue is empty when all information has been read from the list. The associated Summary Message is TRUE, logic 1, if the queue contains some information and FALSE, logic 0, if the queue is empty. Queues can be cleared by reading all the information from the queue. Queues, except the Output Queue, can also be cleared using the \*CLS (clear status) command. A status queue can also be referred to as a Status Register Group.

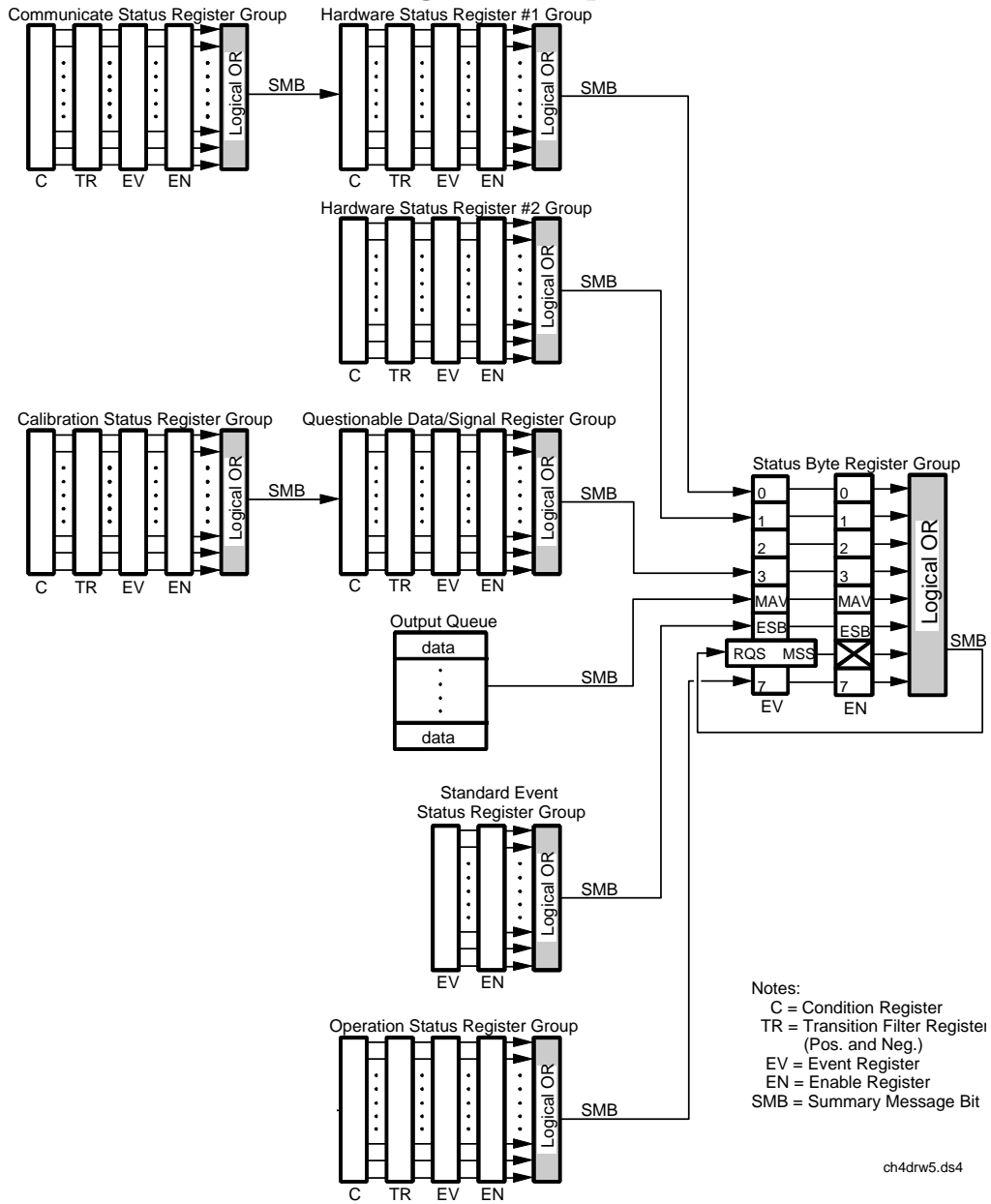
**Figure 8-4**      **Status Data Structure - Queue Mode**



## Status Register Group Contents

**Figure 8-5** shows the Status Register Groups in the Test Set. The contents of each Status Register Group is explained in the following sections.

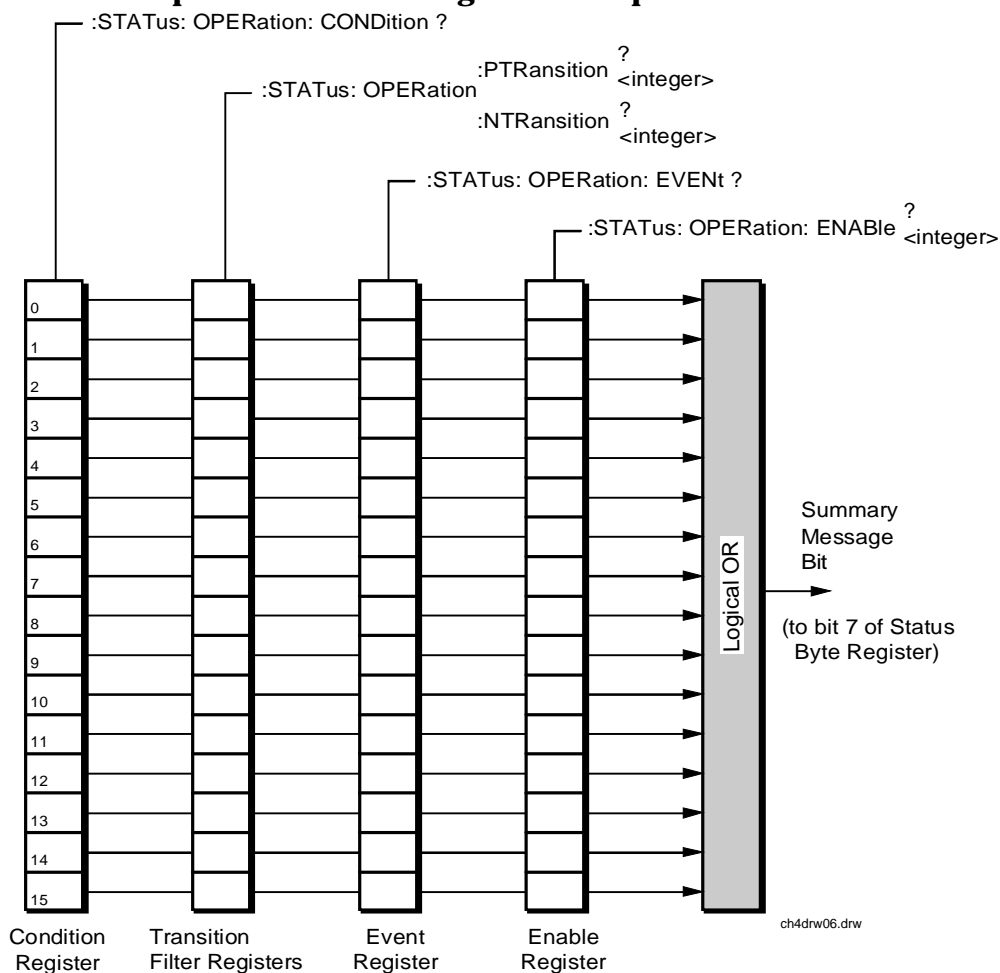
**Figure 8-5 Test Set Status Register Groups**



## Operation Status Register Group

The Operation Status Register Group contains information about the state of the measurement systems in the Test Set. This status group is accessed using the STATus commands. The Operation Status Register Group uses 16-bit registers and includes a Condition Register, Transition Filters, an Event Register, an Enable Register, and a Summary Message. Refer to the "[Status Register Structure Overview](#)" on page 276 for a discussion of status register operation. [Figure 8-6](#) shows the structure and STATus commands for the Operation Status Register Group.

**Figure 8-6** Operation Status Register Group



**Table** shows the Operation Status Register Group Condition Register bit assignments.

### Operation Status Register Group Condition Register Bit Assignments

Bit Number	Binary Weighting	Condition	Comment
15	32768	Not Used (Always 0)	Defined by SCPI Version 1994.0
14	16384	IBASIC Program Running	1 = an IBASIC program is running on the built-in IBASIC controller.
13	8192	Unused in the Test Set	
12	4096	Unused in the Test Set	
11	2048	Unused in the Test Set	
10	1024	Unused in the Test Set	
9	512	Status Register Group Summary Message	1 = one or more of the enabled events have occurred since the last reading or clearing of the Event Register.
8	256	Unused in the Test Set	
7	128	Unused in the Test Set	
6	64	Unused in the Test Set	
5	32	Unused in the Test Set	
4	16	Unused in the Test Set	
3	8	Unused in the Test Set	
2	4	Unused in the Test Set	
1	2	Unused in the Test Set	
0	1	Unused in the Test Set	

### Accessing the Operation Status Register Group's Registers

The following sections show the syntax and give programming examples, using the HP BASIC programming language, for the STATUS commands used to access the Operation Status Register Group's registers.

### **Reading the Condition Register.**

#### **Syntax**

```
STATus:OPERation:CONDition?
```

#### **Example**

```
OUTPUT 714;"STAT:OPER:COND?"  
ENTER 714;Register_value
```

### **Reading the Transition Filters.**

#### **Syntax**

```
STATus:OPERation:PTRansition?  
STATus:OPERation:NTRansition?
```

#### **Example**

```
OUTPUT 714;"STAT:OPER:PTR?"  
ENTER 714;Register_value
```

### **Writing the Transition Filters.**

#### **Syntax**

```
STATus:OPERation:PTRansition <integer>  
STATus:OPERation:NTRansition <integer>
```

#### **Example**

```
OUTPUT 714;"STAT:OPER:PTR 256"
```

### **Reading the Event Register.**

#### **Syntax**

```
STATus:OPERation:EVENT?
```

#### **Example**

```
OUTPUT 714;"STAT:OPER:EVEN?"  
ENTER 714;Register_value
```

### **Clearing the Event Register.**

The EVENT register is cleared whenever it is queried or whenever the Common Command \*CLS is sent to the Test Set.

### **Reading the Enable Register.**

#### **Syntax**

```
STATus:OPERation:ENABle?
```

#### **Example**

```
OUTPUT 714;"STAT:OPER:ENAB?"
```

```
ENTER 714;Register_value
```

### Writing the Enable Register.

#### Syntax

```
STATus:OPERation:ENABle <integer>
```

#### Example

```
OUTPUT 714;"STAT:OPER:ENAB 256"
```

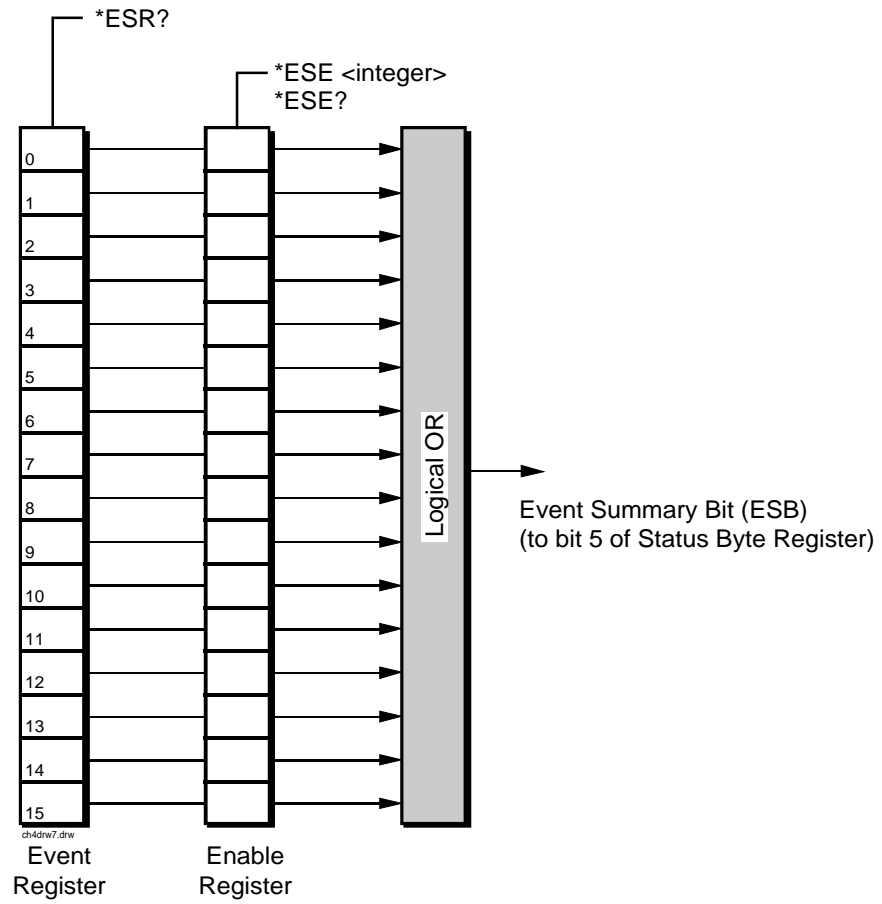
### Clearing the Enable Register.

The ENABLE register is cleared by writing to it with an integer value of zero.

## Standard Event Status Register Group

The Standard Event Status Register Group is a specific implementation of the status register model described in the Status Register Structure Overview section. The conditions monitored by the Standard Event Status Register Group are defined by the IEEE 488.2-1987 Standard. The Standard assigns specific Test Set conditions to specific bits in the Standard Event Status Register. [Table 8-2 on page 287](#) details the Standard Event Status Register bit assignments and their meanings. The Standard Event Status Register Group is accessed using IEEE 488.2 Common Commands. The Standard Event Status Register Group includes an Event Register, an Enable Register, and a Summary Bit. Refer to the "[Status Reporting Structure Overview](#)" on [page 270](#) for a discussion of status register operation. [Figure 8-7](#) shows the structure and IEEE 488.2 Common Commands used to access the Standard Event Status Register Group.

**Figure 8-7** Standard Event Status Register Group





**Table 8-2 Standard Event Status Register Bit Assignments**

<b>Bit Number</b>	<b>Binary Weighting</b>	<b>Condition</b>	<b>Comment</b>
15	32879	Always 0	Reserved by IEEE 488.2
14	16384	Always 0	Reserved by IEEE 488.2
13	8192	Always 0	Reserved by IEEE 488.2
12	4096	Always 0	Reserved by IEEE 488.2
11	2048	Always 0	Reserved by IEEE 488.2
10	1024	Always 0	Reserved by IEEE 488.2
9	512	Always 0	Reserved by IEEE 488.2
8	256	Always 0	Reserved by IEEE 488.2
7	128	Power On	1 = Test Set's power supply has been turned off and then on since the last time this register was read.
6	64	User Request	Not implemented in Test Set.
5	32	Command Error	<p>1 = The Test Set detected an error while trying to process a command. The following events cause a command error:</p> <ul style="list-style-type: none"> <li>• An IEEE 488.2 syntax error. This means that the Test Set received a message that did not follow the syntax defined by the Standard.</li> <li>• A semantic error. For example, the Test Set received an incorrectly spelled command.</li> <li>• The Test Set received a Group Execute Trigger (GET) inside a program message.</li> </ul>

<b>Bit Number</b>	<b>Binary Weighting</b>	<b>Condition</b>	<b>Comment</b>
4	16	Execution Error	<p>1 = The Test Set detected an error while trying to execute a command. The following events cause an execution error:</p> <ul style="list-style-type: none"> <li>• A &lt;PROGRAM DATA&gt; element received in a command is outside the legal range for the Test Set or is inconsistent with the operation of the Test Set.</li> <li>• The Test Set could not execute a valid command due to some Test Set hardware/firmware condition.</li> </ul>
3	8	Device Dependent Error	<p>1 = A Test Set dependent error has occurred. This means that some Test Set operation did not execute properly due to some internal condition, such as overrange. This bit indicates that the error was not a command, query, or execution error.</p>

Bit Number	Binary Weighting	Condition	Comment
2	4	Query Error	<p>1 = An error has occurred while trying to read the Test Set's Output Queue. The following events cause a query error:</p> <ul style="list-style-type: none"> <li>• An attempt is being made to read data from the Output Queue when no data is present or pending.</li> <li>• Data in the Output Queue has been lost. An example of this would be Output Queue overflow.</li> </ul>
1	2	Request Control	<p>1 = The Test Set is requesting permission to become the Active Controller on the HP-IB bus.</p>
0	1	Operation Complete	<p>1 = The Test Set has completed all selected pending operations and is ready to accept new commands. This bit is only generated in response to the *OPC IEEE 488.2 Common Command.</p>

### Accessing the Standard Event Status Register Group's Registers

The following sections show the syntax and give programming examples (using the HP BASIC programming language) for the Common Commands used to access the Standard Event Status Register Group's registers.

## Reading the Event Register.

### Syntax

```
*ESR?
```

### Example

```
OUTPUT 714;"*ESR?"  
ENTER 714;Register_value
```

The \*ESR? query allows the programmer to determine the current contents (bit pattern) of the Standard Event Status Register. The Test Set responds to the \*ESR? query by placing the binary-weighted decimal value of the Standard Event Status Register bit pattern into the Output Queue. The decimal value of the bit pattern will be a positive integer in the range of 0 to 255. The response data is obtained by reading the Output Queue into a numeric variable, integer or real. Reading the Standard Event Status Register clears it (sets all bits to zero).

### Example BASIC program

```
10 INTEGER Std_evn_stat_rg  
20 OUTPUT 714;"*ESR?"  
30 ENTER 714;Std_evn_stat_rg  
40 PRINT Std_evn_stat_rg  
50 END
```

### Example response

```
32
```

## Clearing the Event Register.

The EVENT register is cleared whenever it is queried or whenever the \*CLS Common Command is sent to the Test Set.

## Reading the Enable Register.

### Syntax

```
*ESE?
```

### Example

```
OUTPUT 714;"*ESE?"  
ENTER 714;Register_value
```

The \*ESE? query allows the programmer to determine the current contents (bit pattern) of the Standard Event Status Enable Register. The Test Set responds to the \*ESE? query by placing the binary-weighted decimal value of the Standard Event Status Enable Register bit pattern into the Output Queue. The decimal value of the bit

pattern will be a positive integer in the range of 0 to 255. The response data is obtained by reading the Output Queue into a numeric variable, integer or real.

#### Example BASIC program

```
10 INTEGER Std_evt_enab_rg
20 OUTPUT 714;"ESE?"
30 ENTER 714;Std_evt_enab_rg
40 PRINT Std_evt_enab_rg
50 END
```

#### Example response

```
36
```

### Writing the Enable Register.

#### Syntax

```
*ESE <integer>
```

#### Example

```
OUTPUT 714;"*ESE 255"
```

The \*ESE command sets the bit pattern (bits 0 through 7) of the Standard Event Status Enable Register. The Standard Event Status Enable Register allows the programmer to indicate the occurrence of one or more events (as defined by bits 0 through 7 of the Standard Event Status Register) in bit 5 of the Status Byte Register.

The bit pattern set by the \*ESE command is determined by selecting the desired event(s) from the Standard Event Status Register, setting the value of the bit position(s) to a logical one, setting the value of all non-selected bit positions to a logical zero, and sending the binary-weighted decimal equivalent of bits 0 through 7 after the \*ESE command. For example, if the programmer wished to have the occurrence of a Command Error (bit position 5 in the Standard Event Status Register) and the occurrence of a Query Error (bit position 2 in the Standard Event Status Register) to be reflected in bit 5 of the Status Byte Register, the binary-weighted decimal value of the bit pattern for the Standard Event Status Enable Register would be determined as follows:

**Table 8-3**

Bit Position	7	6	5	4	3	2	1	0	
Logical Value	0	0	1	0	0	1	0	0	
Binary Weighting	128	64	32	16	8	4	2	1	
Decimal Value	0	+0	+32	+0	+0	+4	+0	+0	= 36

**Example**

OUTPUT 714: "\*ESE 36"

The decimal value of the bit pattern must be a positive integer in the range of 0 to 255. Sending a negative number or a number greater than 255 causes an

HP-IB Error: -222 Data out of range.

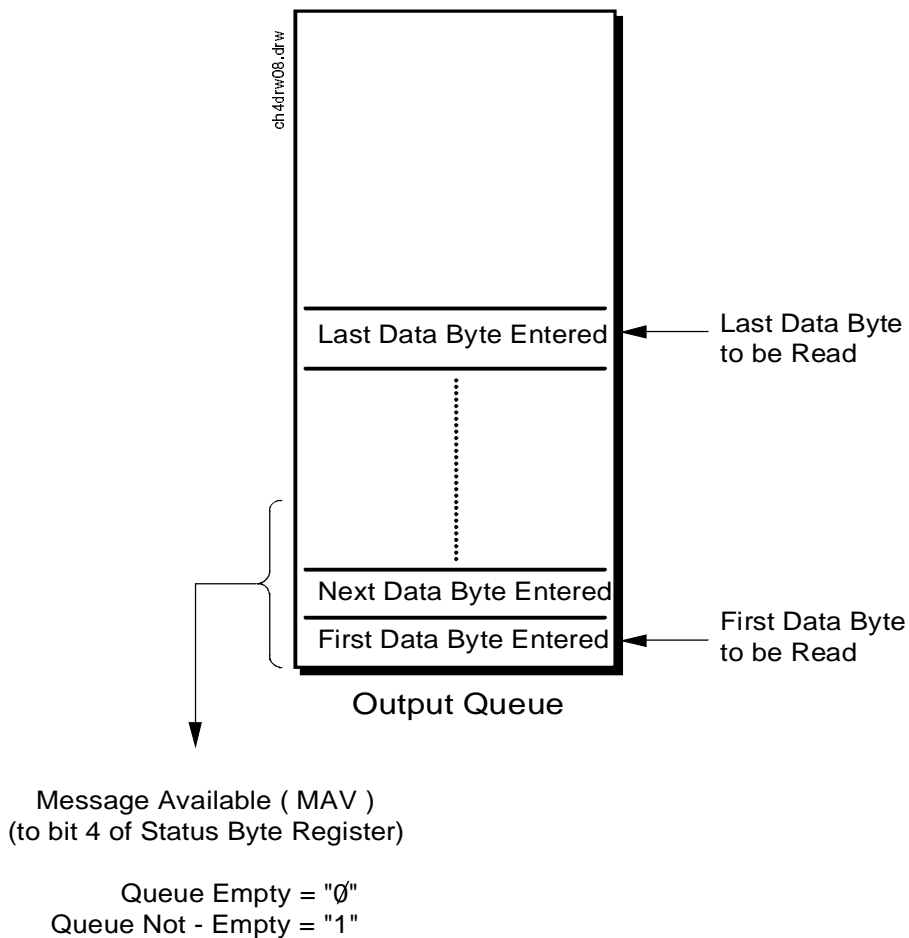
**Clearing the Enable Register.**

The ENABLE register is cleared by writing to it with an integer value of zero.

**Output Queue Group**

The Output Queue Group is a specific implementation of the status queue model described in "[Status Queue Model](#)" on page 279. The Output Queue queue type is defined by the IEEE 488.2-1987 Standard to be a first in, first out (FIFO) queue. The Output Queue Group includes a FIFO queue and a Message Available (MAV) Summary Message. Refer to the "[Status Reporting Structure Overview](#)" on page 270 for an overview of status queue operation. [Figure 8-8](#) shows the structure of the Output Queue Group.

**Figure 8-8**      **Output Queue Group**



## Accessing the Output Queue

When messages are sent to the Test Set, it decodes the message to determine what commands have been sent. Depending upon the type of command, the Test Set's processor sends messages to various parts of the instrument. Many commands generate data which must be sent back to the controller. This data is buffered in the Output Queue until it is read by the controller. The availability of data is summarized in the MAV bit of the Status Byte Register. The state of the MAV message indicates whether or not the Output Queue is empty. The MAV message is TRUE, logic 1, when there is data in the Output Queue and FALSE, logic 0, when it is empty. The Output Queue is read by addressing the Test Set to TALK and then handshaking the bytes out of the Output Queue. Depending upon the type of command sent, the data may appear in the Output Queue almost immediately, or it may take several seconds (as is the case with some Signaling Decoder measurements). Care should be exercised when reading the Output Queue since the HP-IB bus will, by design, wait until the data is available before processing further bus messages.

### Reading the Output Queue.

#### Example

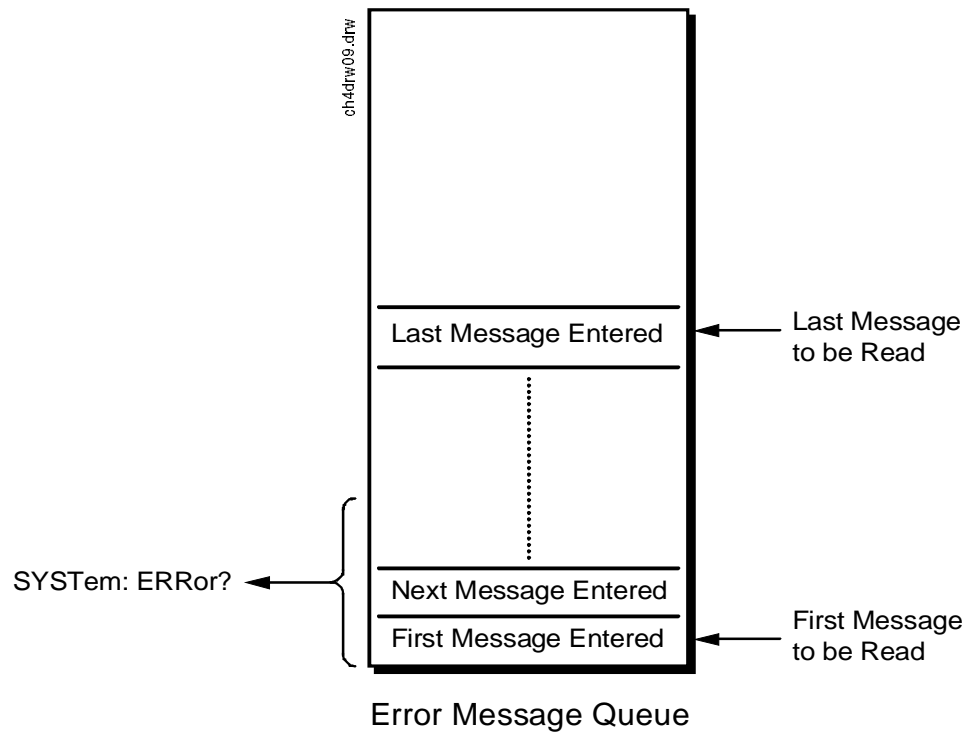
```
Enter 714:Output_data
```

## Error Message Queue Group

The Error Message Queue Group is an implementation of the status queue model described in "[Status Queue Model](#)" on page 279. The Error Message Queue queue type is a first-in, first-out (FIFO) queue that holds up to 20 messages. The Error Message Queue Group includes a FIFO queue but no Message Available (MAV) Summary Message. Refer to the "[Status Reporting Structure Overview](#)" on page 270 for an overview of status queue operation. [Figure 8-9](#) shows the structure of the Error Message Queue Group.



**Figure 8-9**      **Error Message Queue Group**



## Accessing the Error Message Queue

A message appears in the Error Message Queue any time bit 2, 3, 4, or 5 of the Standard Event Status register is asserted. Each message consists of a signed error number, followed by a comma separator, followed by an error description string in double quotes. The maximum length of the error description string is 255 characters. If more than 20 messages are in the queue and another error occurs, the last message is replaced with the message, -350, "Queue overflow". If no messages are in the queue the message, +0, "No error" is returned. Reading a message removes it from the queue. The Error Message Queue is accessed using the SYSTem command. Returned information is read into a numeric variable followed by a string variable.

### Reading the Error Message Queue.

#### Syntax

```
SYSTem:ERRor?
```

#### Example

```
OUTPUT 714;"SYST:ERR?"  
ENTER 714;Error_num,Error_msg$
```

#### Example IBASIC program

```
10 INTEGER Error_num  
20 DIM Error_msg$ [255]  
30 OUTPUT 714;"SYST:ERR?"  
40 ENTER 714;Error_num,Error_msg$  
50 PRINT Error_num;Error_msg$  
60 END
```

#### Example response

```
-113 "Undefined header"
```

## Questionable Data/Signal Register Group

The Questionable Data/Signal Register Group contains information about the quality of the Test Set's output and measurement data. This status group is accessed using the STATus commands. The Questionable Data/Signal Register Group uses 16-bit registers and includes a Condition Register, Transition Filters, an Event Register, an Enable Register, and a Summary Message. Refer to the "[Status Reporting Structure Overview](#)" on page 270 for a discussion of status register operation. [Figure 8-10](#) shows the structure and STATus commands for the Questionable Data/Signal Register Group.

**Figure 8-10 Questionable Data/Signal Register Group**

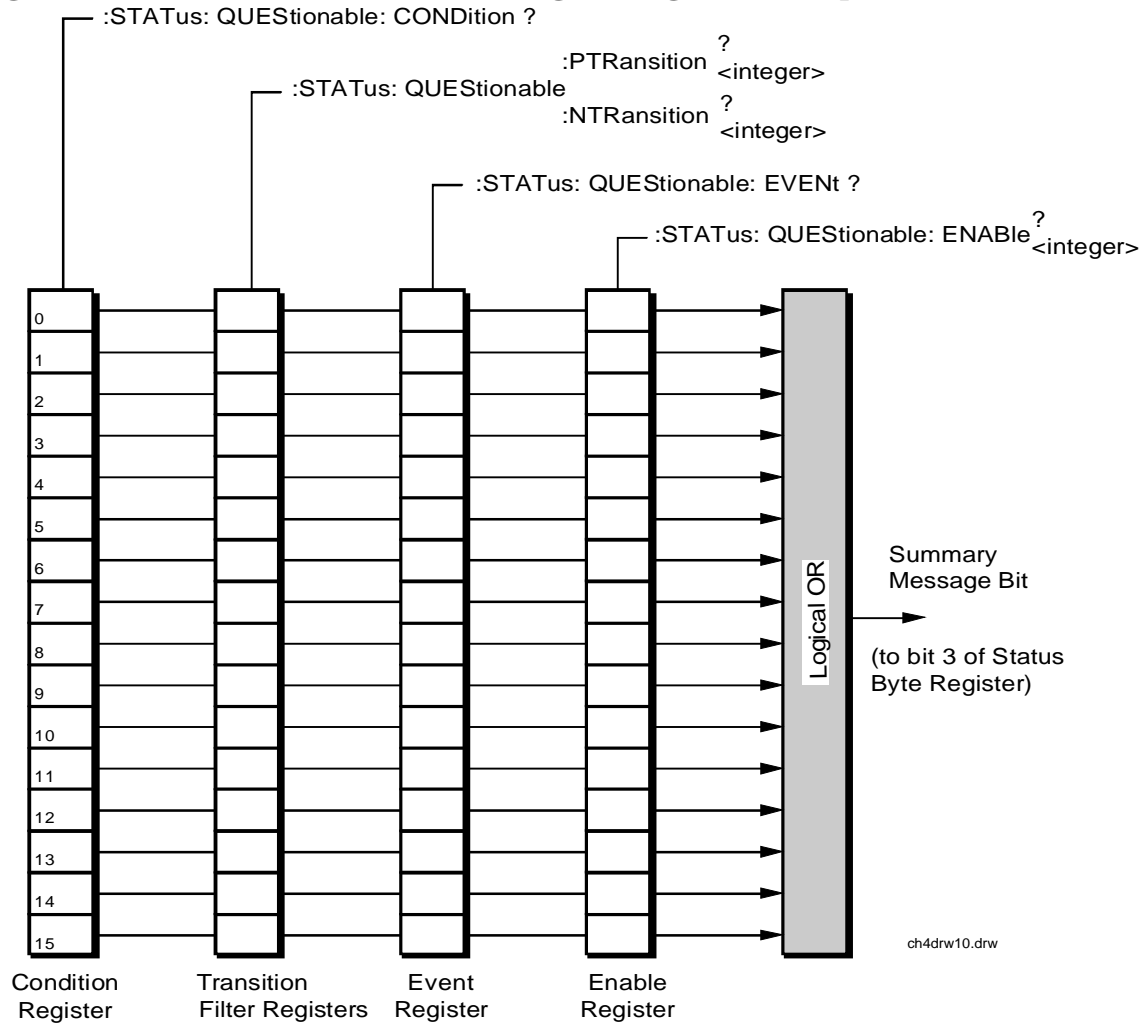


Table 8-4 shows the Questionable Data/Signal Register Group's Condition Register bit assignments.

**Table 8-4 Questionable Data/Signal Register Group, Condition Register Bit Assignments**

<b>Bit Number</b>	<b>Binary Weighting</b>	<b>Condition</b>	<b>Comment</b>
15	32768	Not Used (Always 0)	Defined by SCPI Version 1994.0
14	16384	Unused in Test Set	
13	8192	Unused in Test Set	
12	4096	Unused in Test Set	
11	2048	Unused in Test Set	
10	1024	Unused in Test Set	
9	512	Unused in Test Set	
8	256	Calibration Register Group Summary Message	1 = one or more of the enabled events have occurred since the last reading or clearing of the Event Register.
7	128	Unused in Test Set	
6	64	Unused in Test Set	
5	32	Unused in Test Set	
4	16	Unused in Test Set	
3	8	Unused in Test Set	
2	4	Unused in Test Set	
1	2	Unused in Test Set	
0	1	Unused in Test Set	

## Accessing the Questionable Data/Signal Register Group's Registers

The following sections show the syntax and give programming examples (using the HP BASIC programming language) for the STATUS commands used to access the Questionable Data/Signal Register Group's registers.

### Reading the Condition Register.

#### Syntax

```
STATUS:QUESTIONABLE:CONDITION?
```

#### Example

```
OUTPUT 714;"STAT:QUES:COND?"  
ENTER 714;Register_value
```

### Reading the Transition Filters.

#### Syntax

```
STATUS:QUESTIONABLE:PTRANSITION?  
STATUS:QUESTIONABLE:NTRANSITION?
```

#### Example

```
OUTPUT 714;"STAT:QUES:PTR?"  
ENTER 714;Register_value
```

### Writing the Transition Filters.

#### Syntax

```
STATUS:QUESTIONABLE:PTRANSITION <integer>  
STATUS:QUESTIONABLE:NTRANSITION <integer>
```

#### Example

```
OUTPUT 714;"STAT:QUES:PTR 256"
```

### Reading the Event Register.

#### Syntax

```
STATUS:QUESTIONABLE:EVENT?
```

#### Example

```
OUTPUT 714;"STAT:QUES:EVENT?"  
ENTER 714;Register_value
```

### Clearing the Event Register.

The EVENT register is cleared whenever it is queried or whenever the Common Command \*CLS is sent to the Test Set.

### Reading the Enable Register.

#### Syntax

```
STATus:QUEStionable:ENABle?
```

#### Example

```
OUTPUT 714; "STAT:QUES:ENAB?"  
ENTER 714; Register_value
```

### Writing the Enable Register.

#### Syntax

```
STATus:QUEStionable:ENABle <integer>
```

#### Example

```
OUTPUT 714; "STAT:QUES:ENAB 256"
```

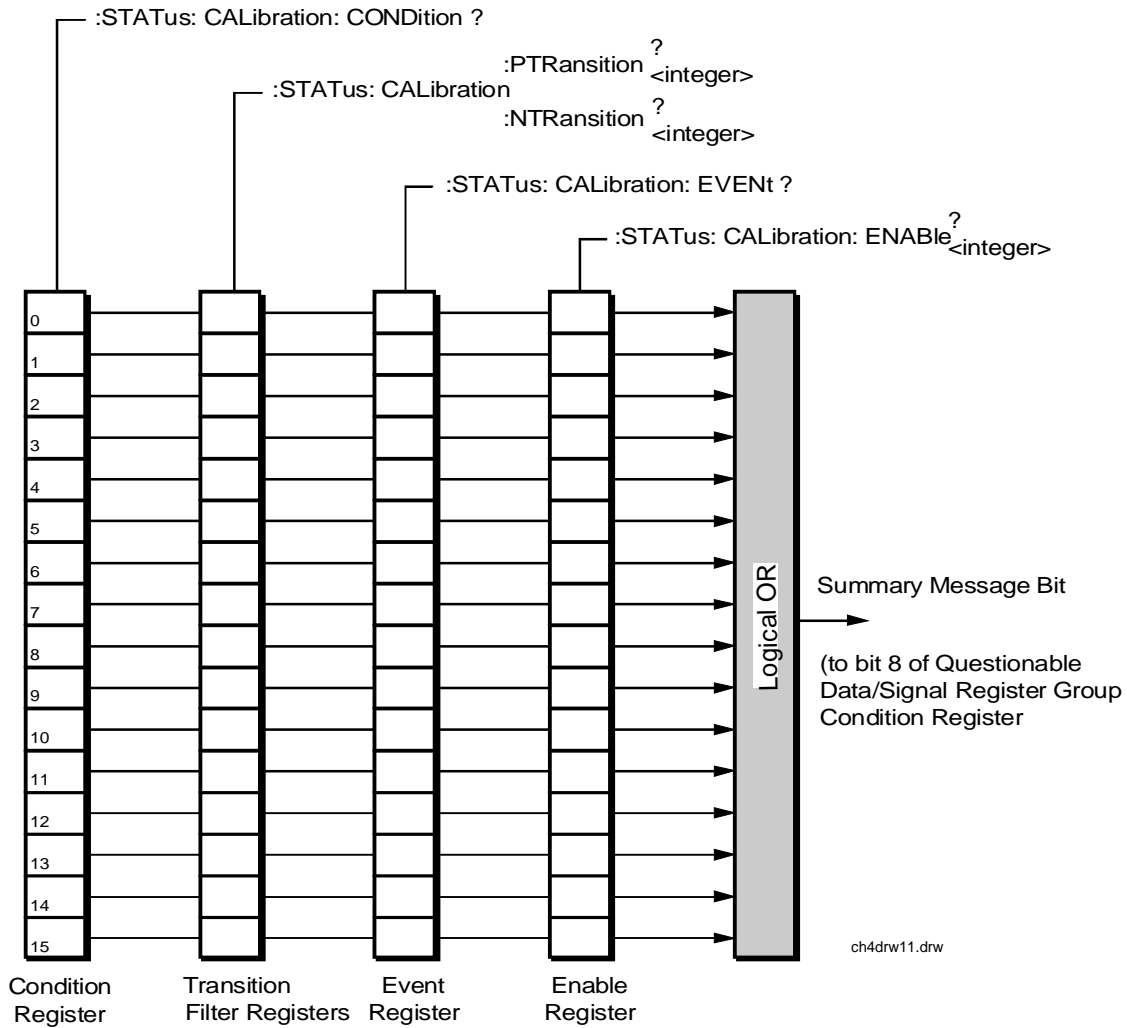
### Clearing the Enable Register.

The ENABLE register is cleared by writing to it with an integer value of zero.

## Calibration Status Register Group

The Calibration Status Register Group contains information about the Test Set's hardware. This status group is accessed using the STATus commands. The Calibration Status Register Group uses 16-bit registers and includes a Condition Register, Transition Filters, an Event Register, an Enable Register, and a Summary Message. Refer to the ["Status Reporting Structure Overview" on page 270](#) for a discussion of status register operation. [Figure 8-11](#) shows the structure and STATus commands for the Calibration Status Register Group.

**Figure 8-11 Calibration Status Register Group**



**Table 8-5** details the Calibration Status Register Group's Condition Register bit assignments.

**Table 8-5 Calibration Status Register Group, Condition Register Bit Assignments**

<b>Bit Number</b>	<b>Binary Weighting</b>	<b>Condition</b>	<b>Comment</b>
15	32768	Not Used (Always 0)	Defined by SCPI Version 1994.0
14	16384	Unused in Test Set	
13	8192	Unused in Test Set	
12	4096	Unused in Test Set	
11	2048	Unused in Test Set	
10	1024	Unused in Test Set	
9	512	Unused in Test Set <sup>a</sup>	
8	256	Unused in Test Set	
7	128	Unused in Test Set	
6	64	Unused in Test Set	
5	32	Unused in Test Set	
4	16	TX Power Auto-Zero Failed	
3	8	Voltmeter Self-Calibration Failed	
2	4	Counter Self-Calibration Failed	
1	2	Sampler Self-Calibration Failed	
0	1	Spectrum Analyzer Self-Calibration Failed	

a. This is a test



## Accessing the Calibration Status Register Group's Registers

The following sections show the syntax and give programming examples (using the HP BASIC programming language) for the STATUS commands used to access the Calibration Status Register Group's registers.

### Reading the Condition Register.

#### Syntax

```
STATUS:CALibration:CONDition?
```

#### Example

```
OUTPUT 714;"STAT:CAL:COND?"  
ENTER 714;Register_value
```

### Reading the Transition Filters.

#### Syntax

```
STATUS:CALibration:PTRansition?  
STATUS:CALibration:NTRansition?
```

#### Example

```
OUTPUT 714;"STAT:CAL:PTR?"  
ENTER 714;Register_value
```

### Writing the Transition Filters.

#### Syntax

```
STATUS:CALibration:PTRansition <integer>  
STATUS:CALibration:NTRansition <integer>
```

#### Example

```
OUTPUT 714;"STAT:CAL:PTR 256"
```

### Reading the Event Register.

#### Syntax

```
STATUS:CALibration:EVENT?
```

#### Example

```
OUTPUT 714;"STAT:CAL:EVENT?"  
ENTER 714;Register_value
```

### **Syntax.**

#### **Clearing the Event Register.**

The EVENT register is cleared whenever it is queried or whenever the Common Command \*CLS is sent to the Test Set.

#### **Reading the Enable Register.**

##### **Syntax**

```
STATus:CALibration:ENABle?
```

##### **Example**

```
OUTPUT 714;"STAT:CAL:ENAB?"  
ENTER 714;Register_value
```

#### **Writing the Enable Register.**

##### **Syntax**

```
STATus:CALibration:ENABle <integer>
```

##### **Example**

```
OUTPUT 714;"STAT:CAL:ENAB 256"
```

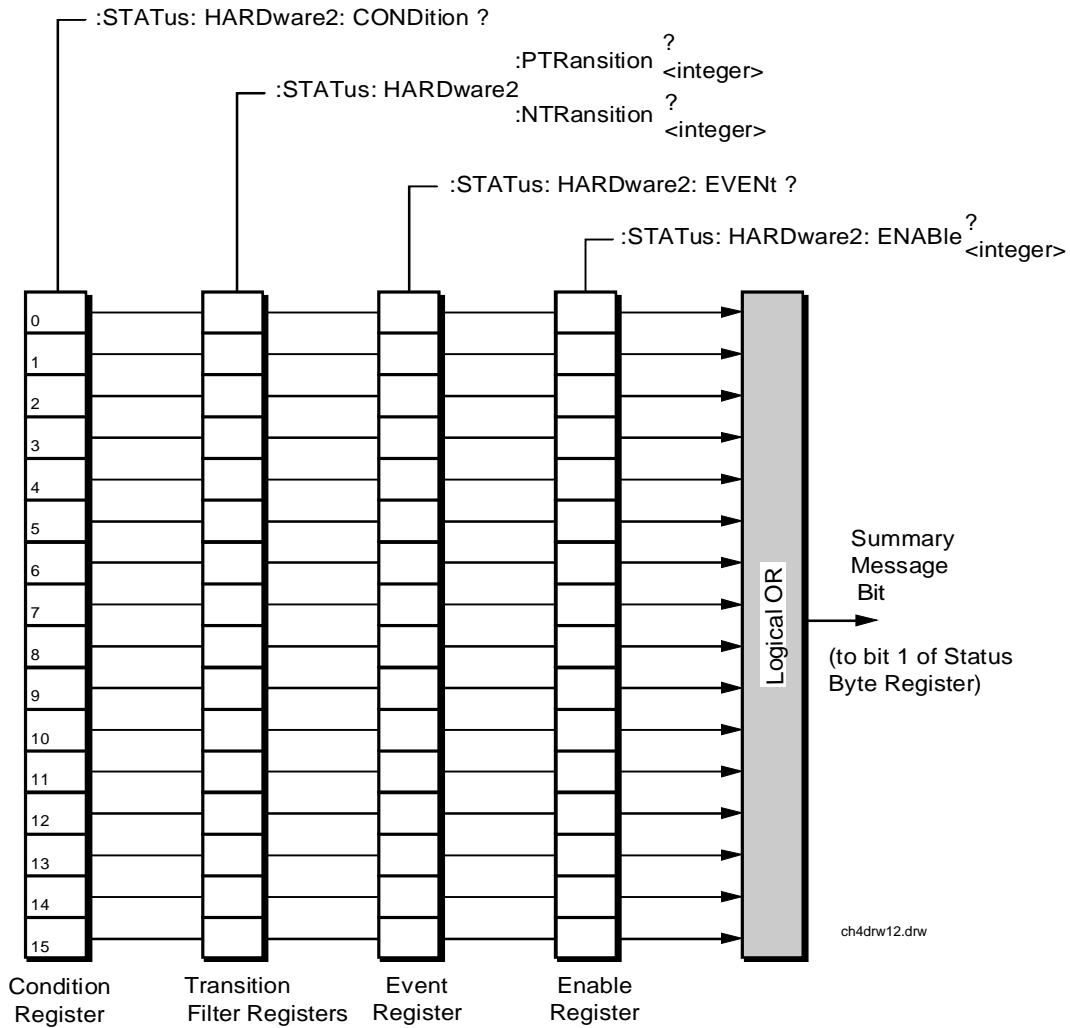
#### **Clearing the Enable Register.**

The ENABLE register is cleared by writing to it with an integer value of zero.

## **Hardware Status Register #2 Group**

The Hardware Status Register #2 Group contains information about the Test Set's hardware. This status group is accessed using the STATus commands. The Hardware Status Register #2 Group uses 16-bit registers and includes a Condition Register, Transition Filters, an Event Register, an Enable Register, and a Summary Message. Refer to the "[Status Reporting Structure Overview](#)" on page 270 for a discussion of status register operation. [Figure 8-12](#) shows the structure and STATus commands for the Hardware Status Register #2 Group.

**Figure 8-12 Hardware Status Register #2 Group**



**Table 8-6** shows the Hardware Status Register Group #2's Condition Register bit assignments.

**Table 8-6 Hardware Status Register Group #2, Condition Register Bit Assignments**

<b>Bit Number</b>	<b>Binary Weighting</b>	<b>Condition</b>	<b>Comment</b>
15	32768	Not Used (Always 0)	Defined by SCPI Version 1994.0
14	16384	Unused in Test Set	
13	8192	Unused in Test Set	
12	4096	Unused in Test Set	
11	2048	Inconsistent ACP Channel Bandwidth and Resolution Bandwidth	
10	1024	ACP Channel Bandwidth or Channel Offset too wide	
9	512	AFGen1 Frequency Exceeds Variable Frequency Notch Filter Range	
8	256	Requested Audio Voltage Too Large for AFGen2	
7	128	Requested FM Deviation Too Large for RF Generator Frequency	
6	64	Requested Simultaneous AM and FM Modulation	Simultaneous AM and FM modulation is not allowed.
5	32	Audio Input Level Auto Ranging Error	
4	16	RF Input Level Auto Ranging Error	
3	8	RF Input Frequency Auto Tuning Error	

Bit Number	Binary Weighting	Condition	Comment
2	4	RF Gen/RF Anl/RF Offset Frequency Combination Not Possible	(RF Gen Tune Freq) - (RF Anal Tune Freq) not equal to (RF Offset Freq)
1	2	RF Generator Amplitude Level Too High for Selected Output Port	
0	1	Spectrum Analyzer Reference Level Too High/Low For Selected Input Port	

### Accessing the Hardware Status Register #2 Group's Registers

The following sections show the syntax and give programming examples (using the HP BASIC programming language) for the STATUS commands used to access the Hardware Status Register #2 Group's registers.

#### Reading the Condition Register.

##### Syntax

```
STATUS:HARDware2:CONDition?
```

##### Example

```
OUTPUT 714;"STAT:HARD2:COND?"
ENTER 714;Register_value
```

#### Reading the Transition Filters.

##### Syntax

```
STATUS:HARDware2:PTRansition?
STATUS:HARDware2:NTRansition?
```

##### Example

```
OUTPUT 714;"STAT:HARD2:PTR?"
ENTER 714;Register_value
```

## Writing the Transition Filters.

### Syntax

```
STATus:HARDware2:PTRansition <integer>  
STATus:HARDware2:NTRansition <integer>
```

### Example

```
OUTPUT 714;"STAT:HARD2:PTR 256"
```

## Reading the Event Register.

### Syntax

```
STATus:HARDware2:EVENT?
```

### Example

```
OUTPUT 714;"STAT:HARD2:EVENT?"  
ENTER 714;Register_value
```

## Clearing the Event Register.

The EVENT register is cleared whenever it is queried or whenever the Common Command \*CLS is sent to the Test Set.

## Reading the Enable Register.

### Syntax

```
STATus:HARDware2:ENABLE?
```

### Example

```
OUTPUT 714;"STAT:HARD2:ENAB?"  
ENTER 714;Register_value
```

## Writing the Enable Register.

### Syntax

```
STATus:HARDware2:ENABLE <integer>
```

### Example

```
OUTPUT 714;"STAT:HARD2:ENAB 256"
```

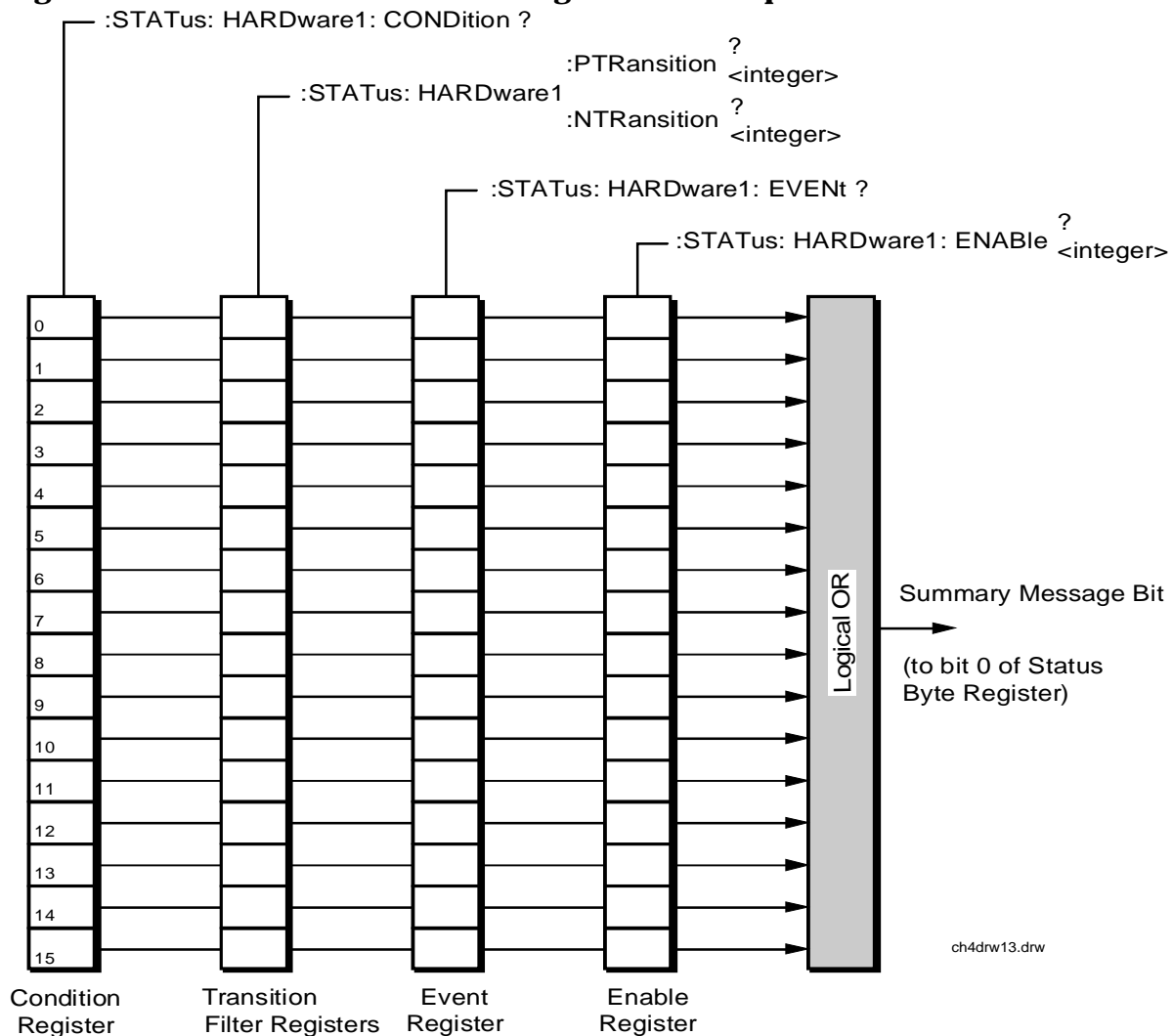
## Clearing the Enable Register.

The ENABLE register is cleared by writing to it with an integer value of zero.

## Hardware Status Register #1 Group

The Hardware Status Register #1 Group contains information about the Test Set's hardware. This status group is accessed using the STATus commands. The Hardware Status Register #1 Group uses 16-bit registers and includes a Condition Register, Transition Filters, an Event Register, an Enable Register, and a Summary Message. Refer to the "[Status Reporting Structure Overview](#)" on page 270 section for a discussion of status register operation. [Figure 8-13](#) shows the structure and STATus commands for the Hardware Status Register #1 Group.

**Figure 8-13** Hardware Status Register #1 Group



[Table 8-7](#) shows the Hardware Status Register Group #1's Condition Register bit assignments.

**Table 8-7 Hardware Status Register Group #1, Condition Register Bit Assignments**

<b>Bit Number</b>	<b>Binary Weighting</b>	<b>Condition</b>	<b>Comment</b>
15	32768	Not Used (Always 0)	Defined by SCPI Version 1994.0
14	16384	Radio Interface Card Interrupt #2 Tripped	
13	8192	Radio Interface Card Interrupt #1 Tripped	
12	4096	Signaling Decoder Measurement Results Available	
11	2048	Signaling Decoder Input Level Too Low	
10	1024	Signaling Decoder is Measuring	
9	512	Signaling Decoder is Armed	
8	256	Signaling Encoder Sending Auxiliary Information	If the Signaling Mode selected has two information fields, such as the AMPS <i>Filler</i> and <i>Message</i> fields, and both fields are being sent, this bit will be set.
7	128	Signaling Encoder Sending Information	If the Signaling Mode selected has only one information field and the field is being sent, this bit will be set high. If the Signaling Mode selected has two information fields, such as the AMPS <i>Filler</i> and <i>Message</i> fields, and only one field is being sent, this bit will be set high. This bit is not active if the Signaling Encoder Mode is set to Function Generator.
6	64	Communication Register Group Summary Message	1 = one or more of the enabled events have occurred since the last reading or clearing of the Event Register.
5	32	Measurement Limit(s) Exceeded	This bit is set high if the Measurement High Limit or Low Limit is exceeded.
4	16	Power-up Self Test(s) Failed	
3	8	Overpower Protection Tripped	



Bit Number	Binary Weighting	Condition	Comment
2	4	Unused in Test Set	
1	2	External Mike Keyed	
0	1	External Battery Voltage Low	

## Accessing the Hardware Status Register #1 Group's Registers

The following sections show the syntax and give programming examples (using the HP BASIC programming language) for the STATus commands used to access the Hardware Status Register #1 Group's registers.

### Reading the Condition Register.

#### Syntax

```
STATus:HARDware1:CONDition?
```

#### Example

```
OUTPUT 714;"STAT:HARD1:COND?"
ENTER 714;Register_value
```

### Reading the Transition Filters.

#### Syntax

```
STATus:HARDware1:PTRansition?
STATus:HARDware1:NTRansition?
```

#### Example

```
OUTPUT 714;"STAT:HARD1:PTR?"
ENTER 714;Register_value
```

### Writing the Transition Filters.

#### Syntax

```
STATus:HARDware1:PTRansition <integer>
STATus:HARDware1:NTRansition <integer>
```

#### Example

```
OUTPUT 714;"STAT:HARD1:PTR 256"
```

### **Reading the Event Register.**

#### **Syntax**

```
STATus:HARDware1:EVENT?
```

#### **Example**

```
OUTPUT 714;"STAT:HARD1:EVEN?"  
ENTER 714;Register_value
```

### **Clearing the Event Register.**

The EVENT register is cleared whenever it is queried or whenever the Common Command \*CLS is sent to the Test Set.

### **Reading the Enable Register.**

#### **Syntax**

```
STATus:HARDware1:ENABLE?
```

#### **Example**

```
OUTPUT 714;"STAT:HARD1:ENAB?"  
ENTER 714;Register_value
```

### **Writing the Enable Register.**

#### **Syntax**

```
STATus:HARDware1:ENABLE <integer>
```

#### **Example**

```
OUTPUT 714;"STAT:HARD1:ENAB 256"
```

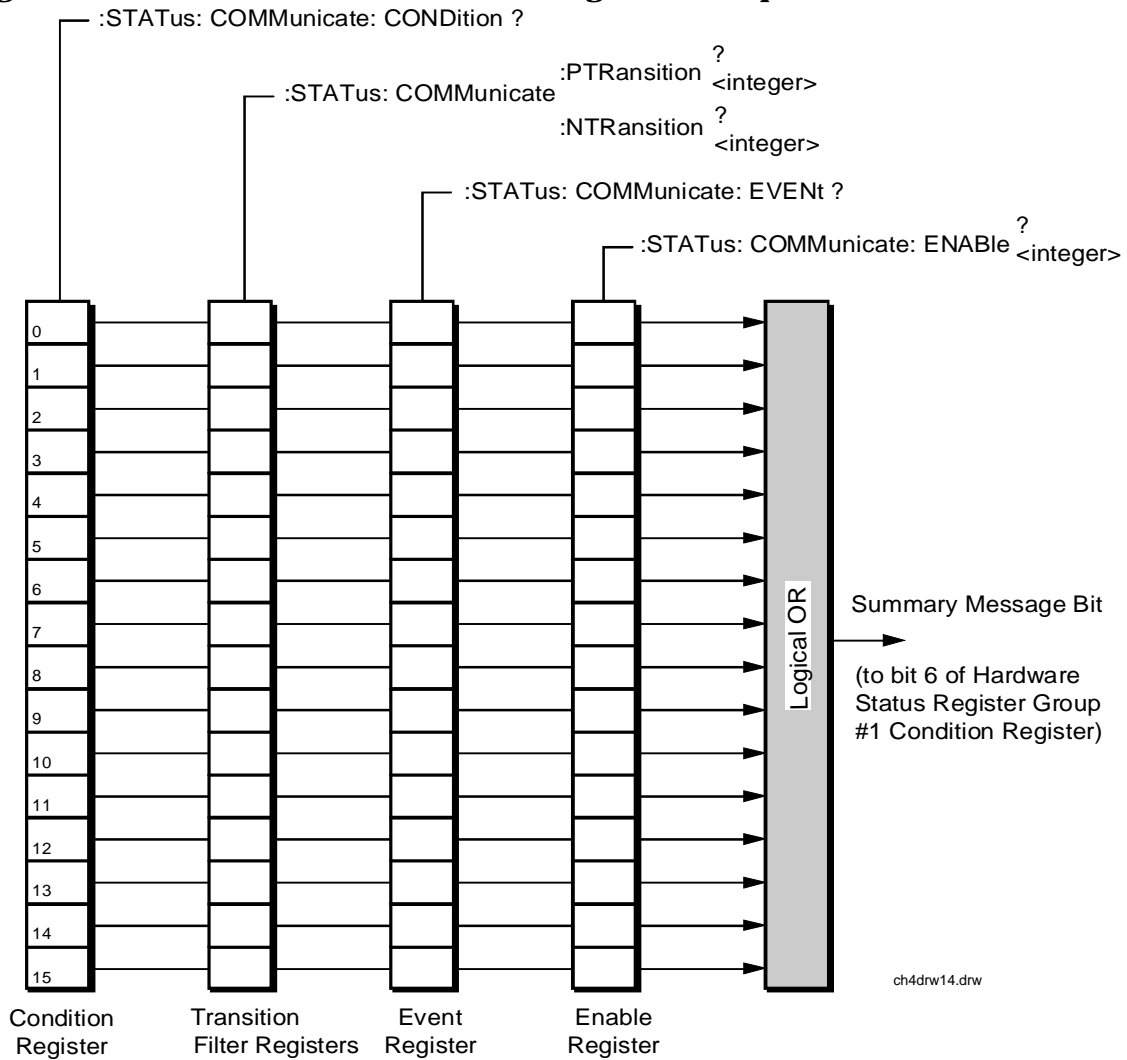
### **Clearing the Enable Register.**

The ENABLE register is cleared by writing to it with an integer value of zero.

## **Communicate Status Register Group**

The Communicate Status Register Group contains information about the Test Set's hardware. This status group is accessed using the STATus commands. The Communicate Status Register Group uses 16-bit registers and includes a Condition Register, Transition Filters, an Event Register, an Enable Register, and a Summary Message. Refer to the "[Status Reporting Structure Overview](#)" on page 270 for a discussion of status register operation. [Figure 8-14](#) shows the structure and STATus commands for the Communicate Status Register Group.

**Figure 8-14 Communicate Status Register Group**



**Table 8-8** shows the Communicate Status Register Group's Condition Register bit assignments.

**Table 8-8 Communicate Status Register Group, Condition Register Bit Assignments**

Bit Number	Binary Weighting	Condition	Comment
15	32768	Not Used (Always 0)	Defined by SCPI Version 1994.0
14	16384	Unused in Test Set	
13	8192	Unused in Test Set	
12	4096	Unused in Test Set	
11	2048	Unused in Test Set	
10	1024	Unused in Test Set	
9	512	Unused in Test Set	
8	256	Unused in Test Set	
7	128	Unused in Test Set	
6	64	Unused in Test Set	
5	32	Unused in Test Set	
4	16	Unused in Test Set	
3	8	Unused in Test Set	
2	4	Unused in Test Set	
1	2	Top Box TX DSP Analyzer Communication Channel Failure	
0	1	Top Box RX DSP Analyzer Communication Channel Failure	

### Accessing the Communicate Status Register Group's Registers

The following sections show the syntax and give programming examples (using the HP BASIC programming language) for the STATus commands used to access the Communicate Status Register Group's registers.

#### Reading the Condition Register.

##### Syntax

STATus:COMMunicate:CONDition?

### Example

```
OUTPUT 714;"STAT:COMM:COND?"  
ENTER 714;Register_value
```

### Reading the Transition Filters.

#### Syntax

```
STATus:COMMunicate:PTRansition?  
STATus:COMMunicate:NTRansition?
```

### Example

```
OUTPUT 714;"STAT:COMM:PTR?"  
ENTER 714;Register_value
```

### Writing the Transition Filters.

#### Syntax

```
STATus:COMMunicate:PTRansition <integer>  
STATus:COMMunicate:NTRansition <integer>
```

### Example

```
OUTPUT 714;"STAT:COMM:PTR 256"
```

### Reading the Event Register.

#### Syntax

```
STATus:COMMunicate:EVENT?
```

### Example

```
OUTPUT 714;"STAT:COMM:EVEN?"  
ENTER 714;Register_value
```

### Clearing the Event Register.

The EVENT register is cleared whenever it is queried or whenever the Common Command \*CLS is sent to the Test Set.

### Reading the Enable Register.

#### Syntax

```
STATus:COMMunicate:ENABle?
```

### Example

```
OUTPUT 714;"STAT:COMM:ENAB?"  
ENTER 714;Register_value
```

### **Writing the Enable Register.**

#### **Syntax**

```
STATUS:COMMunicate:ENABle <integer>
```

#### **Example**

```
OUTPUT 714; "STAT:COMM:ENAB 256"
```

### **Clearing the Enable Register.**

The ENABLE register is cleared by writing to it with an integer value of zero.

## HP-IB Service Requests

The Test Set is capable of generating a “service request” when it requires the Active Controller to take action. Service requests are generally made after the Test Set has completed a task (such as making a measurement) or when an error condition exists (such as an internal self-calibration has failed).

The mechanism by which the Active Controller detects these requests is the SRQ interrupt. Interrupts allow for efficient use of system resources, because the Active Controller may be executing a program until an SRQ interrupt occurs. If SRQ interrupts are enabled in the Active Controller, the occurrence of an interrupt can initiate a program branch to a routine which “services” the interrupt (executes some remedial action). The operating and/or programming manuals for each controller describe the controller’s capability to set up and respond to SRQ interrupts.

This section describes the steps necessary to properly configure the Test Set to request service using the Service Request (SRQ) function.

### Setting Up and Enabling SRQ Interrupts

Test Set status information is maintained in eight register groups. Information in each register group is summarized into a Summary Message. All of the Summary Messages are, in turn, summarized into the Status Byte Register, either directly to specific bit positions in the Status Byte Register as shown in [table 8-9](#), or indirectly through another register group (refer to "[Status Reporting Structure Overview](#)" on [page 270](#) for a detailed discussion of the register groups and status reporting).

**Table 8-9 Status Byte Register Bit Assignments**

<b>Bit Position</b>	<b>Binary Weighting</b>	<b>Assignments</b>
7	128	Operation Status Register Group Summary Message
6	64	Request Service (RQS) message when read by serial poll, <i>or</i> , Master Summary Status (MSS) message when read by STB? command
5	32	Standard Event Status Bit (ESB) Summary Message
4	16	Output Queue Message Available (MAV) Summary Message
3	8	Questionable Data/Signal Register Group Summary Message
2	4	Unused in Test Set
1	2	Hardware #2 Status Register Group Summary Message
0	1	Hardware #1 Status Register Group Summary Message

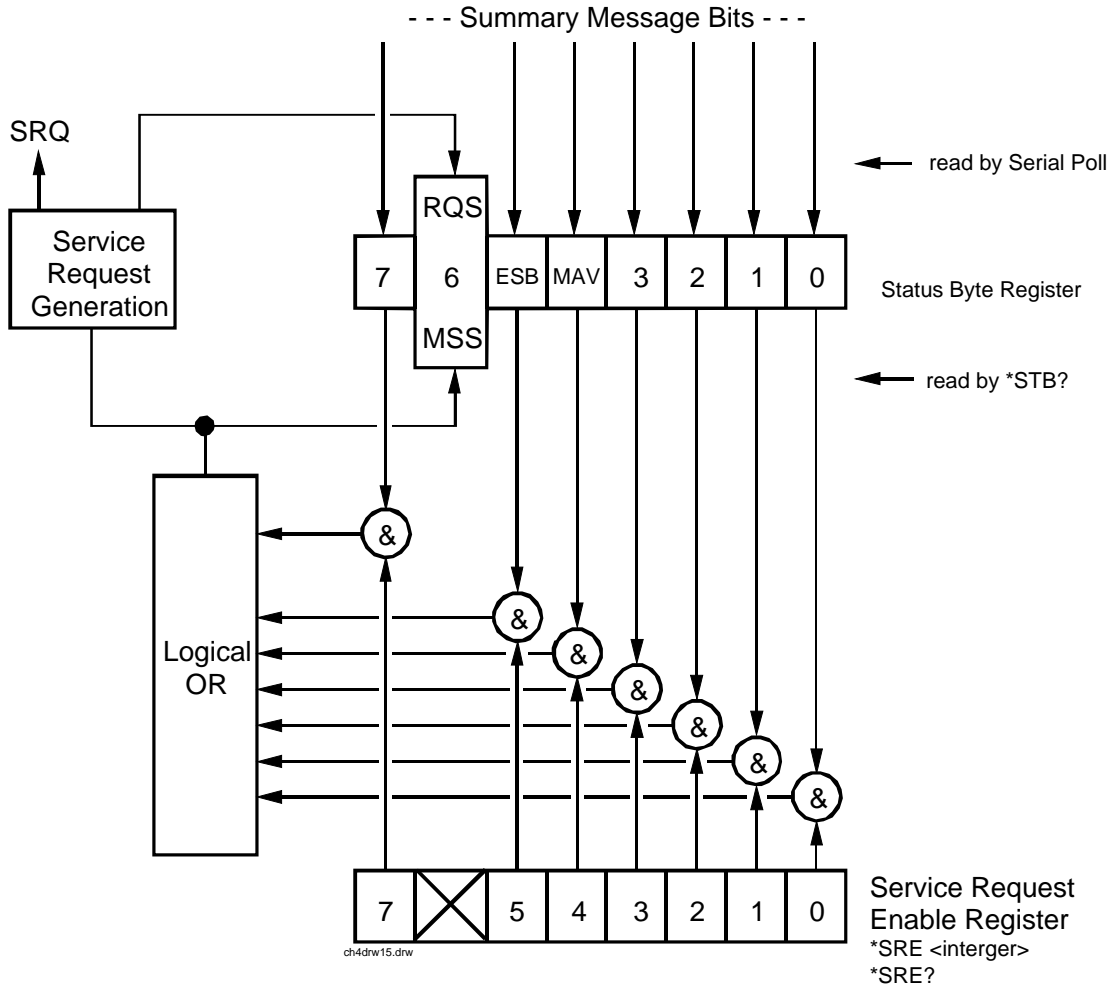
Bits in the Status Byte Register can be used to generate a Service Request (SRQ) message by enabling the associated bit in the Service Request Enable Register. When an enabled service request condition exists, the Test Set sends the Service Request message (SRQ) on the HP-IB bus and reports that it has requested service by setting the Request Service (RQS) bit in the Status Byte register to the TRUE, logic 1, state. When read by a serial poll, the RQS bit is cleared (set to logic 0) so that the RQS message will be FALSE if the Test Set is polled again before a new reason for requesting service has occurred.

## Service Request Enable Register

Service request enabling allows the application programmer to select which Summary Messages in the Status Byte Register may cause a service request. The Service Request Enable Register, illustrated in [figure 8-15](#), is an 8-bit register that enables corresponding Summary Messages in the Status Byte Register.



**Figure 8-15 Service Request Enable Register**



## Reading the Service Request Enable Register

The Service Request Enable Register is read with the \*SRE? Common Command. The \*SRE? query allows the programmer to determine the current contents (bit pattern) of the Service Request Enable Register. The Test Set responds to the \*SRE? query by placing the binary-weighted decimal value of the Service Request Enable Register bit pattern into the Output Queue. The decimal value of the bit pattern will be a positive integer in the range 0 to 255. The response data is obtained by reading the Output Queue into a numeric variable, integer or real.

### Example BASIC program

```
10 INTEGER Srv_rqs_enab_rg
20 OUTPUT 714;"*SRE?"
30 ENTER 714;Srv_rqs_enab_rg
40 PRINT Srv_rqs_enab_rg
50 END
```

### Example response

```
18
```

## Writing the Service Request Enable Register

The Service Request Enable Register is written with the \*SRE Common Command. The \*SRE command sets the bit pattern (bits 0-5 and 7) of the Service Request Enable Register. The Service Request Enable Register allows the programmer to select which condition(s), as defined by bits 0-5 and 7 of the Status Byte Register, will generate a Service Request on the HP-IB bus. The Test Set always ignores bit 6 (binary weight 64) of the bit pattern set by the \*SRE command.

The bit pattern set by the \*SRE command is determined by selecting the desired condition(s) from the Status Byte Register, setting the value of the bit position(s) to a logical one, setting the value of all non-selected bit positions to a logical zero, and sending the binary-weighted decimal equivalent of bits 0-5 and 7 after the \*SRE command. For example, if the programmer wished to have the occurrence of a message available in the Output Queue (bit position 4 in the Status Byte Register) and the occurrence of a condition in the Hardware# 2 Status Register (bit position 1 in the Status Byte Register) to generate a Service Request on the HP-IB bus, the binary-weighted decimal value of the bit pattern for the Service Request Enable Register would be determined as shown in [table 8-10](#).

**Table 8-10 Determining the Service Request Enable Register Bit Pattern**

Bit Position	7	6	5	4	3	2	1	0	
Logical Value	0	X	0	1	0	0	1	0	X = ignored by the Test Set
Binary Weighting	128	X	32	16	8	4	2	1	X = ignored by the Test Set
Decimal Value	0+	0+	0+	16+	0+	0+	2+	0	= 18

**Example**

OUTPUT 714; "\*SRE 18"

**NOTE**

The decimal value of the bit pattern must be a positive integer in the range of 0 to 255. Sending a negative number or a number greater than 255 causes an HP-IB Error: -222 Data out of range.

**Clearing the Service Request Enable Register**

The Service Request Enable Register is cleared by sending the \*SRE Common Command with a decimal value of zero. Clearing the Service Request Enable Register turns off service requests.

**Procedure for Generating a Service Request**

The following steps outline a generalized procedure for properly setting up the Test Set to generate a Service Request (SRQ) message to the Active Controller. This procedure does not include instructions for setting up the Active Controller to respond to the Service Request message generated by the Test Set. Refer to the operating and/or programming manuals for each controller for information describing the controller's capability to set up and respond to SRQ interrupts.

- For register groups with Condition Registers and Transition Filters start at step 1.
  - For register groups with no Condition Register or Transition Filters start at step 5.
1. Determine which conditions, as defined by their bit positions in the Register Group Condition Register, should cause the Summary Message to be set TRUE if they occur.
  2. Determine the polarity of the bit-state transition which will indicate that the condition has occurred.
  3. Set the Register Group Transition Filters to the correct polarity to pass the bit-state transition to the Event Register.
  4. Go to step 6.

5. Determine which conditions, as defined by their bit positions in the Register Group Event Register, should cause the Summary Message to be set TRUE if they occur.
6. Set the correct bits in the Register Group Enable Register to generate the Summary Message if the condition has been latched into the Register Group Event Register.
7. If the Summary Message is a bit in a Register Group that is not the Status Byte Register go to step 1.
8. Set the correct bits in the Service Request Enable Register for all Register Group Summary Messages selected in steps 1 through 6.

## Example BASIC Program to Set Up and Service an SRQ Interrupt

The following HP BASIC program was written for an HP 9000 Series 300 Controller and a Test Set. The program assumes that the Test Set is the only instrument on the bus. The program sets up an interrupt from the Standard Event Status Register Group, the Calibration Status Register Group, and the Hardware Status Register #1 Group. For demonstration purposes the program is written to stay in a dummy loop waiting for an interrupt from the Test Set

### Example

```
10 OPTION BASE 1
20 COM/IO_names/INTEGER Inst_address,Std_event_reg,Calibration_reg
30 COM /IO_names/ INTEGER
Hardware1_reg,Srq_enab_reg,Status_byte,Event_reg
40 !
50 ! Define instrument address
60 Inst_address=714
70 !
80 PRINTER IS CRT
90 CLEAR SCREEN
100 !
110 ! Reset the Test Set to bring it to a known state
120 OUTPUT Inst_address;"*RST"
130 !
140 ! Clear the Test Set status reporting system
150 OUTPUT Inst_address;"*CLS"
160 !
170 ! Set up the desired interrupt conditions in the Test Set:
180 !
190 ! 1) Standard Event Status Register Group
200 ! Event register conditions which will set the Summary Message
210 ! TRUE if they occur:
220 ! Bit 5: Command Error decimal value = 2^5 = 32
230 ! Bit 4: Execution Error decimal value = 2^4 = 16
240 ! Bit 3: Device Dependent Error decimal value = 2^3 = 8
250 ! Bit 2: Query Error decimal value = 2^2 = 4
260 !
270 Std_event_reg=32+16+8+4
280 !
290 ! Set up the Standard Event Status Enable Register to generate
the
300 ! Summary Message
310 !
320 OUTPUT Inst_address;"*ESE";Std_event_reg
330 !
340 ! 2) Calibration Status Register Group
```

```

350 ! Condition register conditions which will set the Summary
Message
360 ! TRUE if they occur:
370 ! Bit 4: TX Auto-zero failed decimal value = 2^4 = 16
380 ! Bit 3: Voltmeter Self-cal failed decimal value = 2^3 = 8
390 ! Bit 2: Counter Self-cal failed decimal value = 2^2 = 4
400 ! Bit 1: Sampler Self_cal failed decimal value = 2^1 = 2
410 ! Bit 0: Spec Anal Self-cal failed decimal value = 2^0 = 1
420 !
430 Calibration_reg=16+8+4+2+1
440 !
450 ! Set the Transition Filters to allow only positive transitions
in
460 ! the assigned condition(s) to pass to the Event Register
470 !
480 OUTPUT Inst_address;"STAT:CAL:PTR";Calibration_reg
490 OUTPUT Inst_address;"STAT:CAL:NTR 0"
500 !

510 ! Set up the Calibration Status Register Group Enable Register to
520 ! generate the Summary Message.
530 !
540 OUTPUT Inst_address;"STAT:CAL:ENAB";Calibration_reg
550 !
560 ! The Calibration Status Register Group Summary Message is passed
to
570 ! the Status Byte Register through Bit 8 in the Questionable
580 ! Data/Signal Register Group Condition Register. The Questionable
590 ! Data/Signal Register Group must be configured to set its Summary
600 ! Message TRUE if the Summary Message from the Calibration Status
610 ! Register Group is TRUE. Therefore Bit 8 (2^8=256) in the
Questionable
620 ! Data/Signal Register Group Enable Register must be set HIGH.
630 !
640 OUTPUT Inst_address;"STAT:QUES:ENAB 256"
650 !
660 ! 3) Hardware Status Register #1 Group
670 ! Condition register conditions which will set the Summary
Message
680 ! TRUE if they occur:
690 ! Bit 5: Measurement limits exceeded decimal value = 2^5 =
32
700 ! Bit 4: Power-up Self-test failed decimal value = 2^4 = 16
710 ! Bit 3: Overpower protection tripped decimal value = 2^3 = 8
720 !
730 Hardware1_reg=32+16+8
740 !
750 ! Set the Transition Filters to allow only positive transitions
in
760 ! the assigned condition(s) to pass to the Event Register
770 !
780 OUTPUT Inst_address;"STAT:HARD1:PTR";Hardware1_reg
790 OUTPUT Inst_address;"STAT:HARD1:NTR 0"
800 !
810 ! Set up the Hardware Status Register #1 Group Enable Register
to
820 ! generate the Summary Message.
830 !
840 OUTPUT Inst_address;"STAT:HARD1:ENAB";Hardware1_reg
850 !
860 ! 4) Set the correct Summary Message bit(s) in the Service Request
870 ! Enable Register to generate a Service Request (SRQ) if the
880 ! Summary Message(s) become TRUE.
890 ! Bit 5 = Standard Event Status Register Summary Message
900 ! decimal value = 2^5
= 32
910 ! Bit 3 = Questionable Data/Signal Register Group Summary
Message
920 ! decimal value = 2^3
= 8
930 ! Bit 0 = Hardware Status Register #1 Group Summary Message
940 ! decimal value = 2^0
= 1
950 !
960 Srq_enab_reg=32+8+1

```

```
970 OUTPUT Inst_address;"*SRE";Srq_enab_reg
980 !
990 ! 5) Set up the Active Controller to respond to an SRQ interrupt:
1000 ! Call subprogram Check_interrupt if an SRQ condition exists on
select
1010 ! code 7. The interrupt priority level is set to 15 (highest
level).
1020 !
1030 ON INTR 7,15 CALL Srvic_e_interupt
1040 !
1050 ! 6) Enable interrupts on select code 7:
1060 ! The interface mask is set to a value of 2 which enables
interrupts on
1070 ! the HP-IB bus when the SRQ line is asserted.
1080 !
1090 ENABLE INTR 7;2
1100 !
1110 ! Start of the dummy loop:
1120 !
1130 LOOP
1140 DISP "I am sitting in a dummy loop."
1150 END LOOP
1160 !
1170 END
1180 !
1190 Srvic_e_interupt:SUB Srvic_e_interupt
1200 !
1210 OPTION BASE 1
1220 COM /Io_names/ INTEGER Inst_address,Std_event_reg,Calibration_reg
1230 COM / Io_names/ INTEGER
Hardware1_reg,Srq_enab_reg,Status_byte,Event_reg
1240 !
1250 !Turn off interrupts while processing the current interrupt.
1260 OFF INTR 7
1270 !
1280 !Conduct a SERIAL POLL to read the Status Byte and clear the SRQ:
1290 !
1300 Status_byte=SPOLL(Inst_address)
1310 !
1320 ! Determine which Register Group(s) caused the interrupt. Since
three
1330 ! were enabled, all three must be checked:
1340 !
1350 IF BIT(Status_byte,5) THEN GOSUB Srvic_std_evnt
1360 IF BIT(Status_byte,3) THEN GOSUB Srvic_calib
1370 IF BIT(Status_byte,0) THEN GOSUB Srvic_hard1
1380 !
1390 ! Re-enable the interrupt before leaving the service routine
1400 !
1410 ENABLE INTR 7;2
1420SUBEXIT
1430!
1440 Srvic_std_evnt:!
1450 ! This routine would determine which bit(s) in the Standard Event
1460 ! Status Register are TRUE, logic 1, and take appropriate action.
1470 ! NOTE: Read the Event Register to clear it. If the Event
Register is
1480 ! not cleared it will NOT latch another event, thereby preventing
1490 ! the Test Set from generating another SRQ.
1500 !
1510 OUTPUT Inst_address;"*ESR?"
1520ENTER Inst_address;Event_reg
1530RETURN
1540!
1550 Service_calib:!
1560! This routine would determine which bit(s) in the Calibration
Status
1570! Register Group Event Register are TRUE, logic 1, and take
1580! appropriate action.
1590! NOTE: Read the Event Register to clear it. If the Event Register
is
1600! not cleared it will NOT latch another event from the Condition
1610! Register, thereby preventing the Test Set from generating another
SRQ.
1620!
```

```
1630OUTPUT Inst_address;"STAT:CAL:EVEN?"
1640ENTER Inst_address;Event_reg
1650RETURN
1660

!1670 Srvce_hard1:!
1680 ! This routine would determine which bit(s) in the Hardware
Status
1690 ! Register #1 Group Event Register are TRUE, logic 1, and take
1700 ! appropriate action.
1710 ! NOTE: Read the Event Register to clear it. If the Event
Register is
1720 ! not cleared it will NOT latch another event from the Condition
1730 ! Register, thereby preventing the Test Set from generating
another SRQ.
1740 !
1750 OUTPUT Inst_address;"STAT:HARD1:EVEN?"
1760 ENTER Inst_address;Event_reg
1770 RETURN
1780 !
1790 SUBEND
```

## Instrument Initialization

This section discusses the various methods available to the programmer to initialize the Test Set to a known state.

With over 22 instruments utilizing greater than 25 screens containing hundreds of fields which can be programmed through the HP-IB bus, a hard copy list of the default condition for every field in every instrument screen would be cumbersome. The recommended method of determining the default condition for every field in a particular instrument screen is to select the PRESET key, display the instrument screen of interest and view the contents of the fields.

Apart from the individual instruments it is important, from a programmatic perspective, to know the default conditions of the I/O configuration of the Test Set and how it may be affected by the various methods of initialization. Seven screens are used to control the I/O configuration of the Test Set:

- CONFIGURE screen
- I/O CONFIGURE screen
- PRINT CONFIGURE screen
- TESTS (Main Menu) screen
- TESTS (Execution Conditions) screen
- TESTS ( External Devices) screen
- TESTS (Printer Setup) screen

The following sections discuss how the various methods of initialization affect these seven screens as well as the Status Reporting Structure of the Test Set.



## Methods of Initialization

There are six methods of initializing the Test Set:

1. Power On Reset
2. Front panel PRESET key
3. \*RST IEEE 488.2 Common Command
4. Device Clear (DCL) HP-IB Bus Command
5. Selected Device Clear (SDC) HP-IB Bus Command
6. Interface Clear (IFC) HP-IB Bus Command

When the Test Set is initialized some fields are “restored” (put back to their default state), some fields are “maintained” (kept at their current state or value), and some fields are “initialized” (returned to their default value).

The following sections discuss the effects each of the six initialization methods has on the Test Set.

## Power-On Reset

The Power-On Reset is accomplished by applying or cycling AC/DC power to the Test Set.

For the CONFIGURE, PRINT CONFIGURE, TESTS (Execution Conditions), TESTS (Printer Setup) and I/O CONFIGURE screens, [table 8-11](#) lists the fields which are restored/initialized when the Test Set AC/DC power is cycled. The restored state or initialized value is listed below the field name. Fields which are not listed are maintained at their current value, whatever that may happen to be. All fields in the TESTS (Main Menu) screen and the TESTS (External Devices) screen are maintained at their current state/value. The current state/value of the maintained fields can be ascertained programmatically.

**Table 8-11** Screen Fields Restored/Initialized During Power-On Reset (Continued)

<b>Instrument CONFIGURE Screen Fields</b>	<b>PRINTER CONFIGURE Screen Fields</b>	<b>TESTS (Execution Conditions) Screen Fields</b>	<b>TESTS (Printer Setup) Screen Fields</b>	<b>I/O CONFIGURE</b>
	Print Title field is cleared.	Test output location:  Crt	Test output location:  Crt	Save/Recall Internal
RF Offset  Off		Results output:  All	Results output:  All	
(Gen)-(Anl) Not displayed		If Unit Under Test Fails:  Continue		
Range Hold  Auto State		Test Procedure run mode:  Continuous		
Notch Coupl None				
RF Display  Chan				
RF Chan Std  North America PCS				

<b>Instrument CONFIGURE Screen Fields</b>	<b>PRINTER CONFIGURE Screen Fields</b>	<b>TESTS (Execution Conditions) Screen Fields</b>	<b>TESTS (Printer Setup) Screen Fields</b>	<b>I/O CONFIGURE</b>
RF Level Offset Off				
RF In/Out 0.0				
Duplex Out 0.0				
Antenna In 0.0				

The Power-On Reset condition in the Test Set was specifically designed to configure the instruments for manual testing of an FM radio. The Power-On Reset default display screen is the RX TEST screen. Other operational characteristics are also affected by the Power-On Reset as follows:

- The Power-up self-test diagnostics are performed.
- The Contents of the SAVE/RECALL registers are not affected.
- All pending operations are aborted.
- Measurement triggering is set to TRIG:MODE:SETT FULL;RETR REP
- All Enable registers are cleared: Service Request, Standard Event, Communicate, Hardware #1, Hardware #2, Operation, Calibration, and Questionable Data/Signal.
- All Negative Transition Filter registers are initialized to all zeros: Communicate, Hardware #1, Hardware #2, Operation, Calibration, and Questionable Data/Signal.
- All Positive Transition Filter registers are initialized to all ones: Communicate, Hardware #1, Hardware #2, Operation, Calibration, and Questionable Data/Signal.
- Any previously received Operation Complete command (\*OPC) is cleared.
- Any previously received Operation Complete query command (\*OPC?) is cleared.

- Calibration data is not affected.
- The HP-IB interface is reset (any pending Service Request is cleared.)
- The contents of the RAM memory are unaffected.
- The Test Set's display screen is in the UNLOCKED state.

## Front-panel PRESET Key

The Front-panel Reset is accomplished by pressing the PRESET key on the front panel of the Test Set.

For the CONFIGURE, PRINT CONFIGURE, TESTS (Execution Conditions), TESTS (Printer Setup) and I/O CONFIGURE screens, [table 8-12](#) lists the fields which are restored/initialized when the front-panel PRESET key is pressed. The restored state or initialized value is listed below the field name. Fields which are not listed are maintained at their current value, whatever that may happen to be. All fields in the TESTS (Main Menu) screen and the TESTS (External Devices) screen are maintained at their current state/value. The current state/value of the maintained fields can be ascertained programmatically.

**Table 8-12** Screen Fields Restored/Initialized During Front Panel Reset (Continued)

<b>Instrument CONFIGURE Screen Fields</b>	<b>PRINTER CONFIGURE Screen Fields</b>	<b>TESTS (Execution Conditions) Screen Fields</b>	<b>TESTS (Printer Setup) Screen Fields</b>	<b>I/O CONFIGURE</b>
	Print Title field is cleared.	Test output location: Crt	Test output location: Crt	Save/Recall Internal
RF Offset Off		Results output: All	Results output: All	
(Gen)-(Anl) Not displayed		If Unit Under Test Fails: Continue		
Range Hold Auto State		Test Procedure run mode: Continuous		
Notch Coupl None				

<b>Instrument CONFIGURE Screen Fields</b>	<b>PRINTER CONFIGURE Screen Fields</b>	<b>TESTS (Execution Conditions) Screen Fields</b>	<b>TESTS (Printer Setup) Screen Fields</b>	<b>I/O CONFIGURE</b>
RF Display Chan				
RF Chan Std North America PCS				
RF Level Offset Off				
RF In/Out 0.0				
Duplex Out 0.0				
Antenna In 0.0				

The Front-panel Reset condition in the Test Set was specifically designed to configure the instruments for manual testing of an FM radio. The Front-panel Reset default display screen is the RX TEST screen. Other operational characteristics are also affected by the Front-panel Reset as follows:

- All pending operations are aborted.
- Measurement triggering is set to TRIG:MODE:SETT FULL;RETR REP.
- Any previously received Operation Complete command (\*OPC) is cleared.
- Any previously received Operation Complete query command (\*OPC?) is cleared.
- The Test Set's display screen is in the UNLOCKED state.
- The Power-up self-test diagnostics are not performed.
- The HP-IB interface is not reset (any pending Service Request is not cleared.)

- The Contents of the SAVE/RECALL registers are not affected.
- Calibration data is not affected.
- All Enable registers are unaffected: Service Request, Standard Event, Communicate, Hardware #1, Hardware #2, Operation, Calibration, and Questionable Data/Signal.
- All Negative Transition Filter registers are unaffected: Communicate, Hardware #1, Hardware #2, Operation, Calibration, and Questionable Data/Signal.
- All Positive Transition Filter registers are unaffected: Communicate, Hardware #1, Hardware #2, Operation, Calibration, and Questionable Data/Signal.
- The contents of the RAM memory are unaffected.

### \*RST IEEE 488.2 Common Command

The \*RST Reset is accomplished by sending the \*RST Common Command to the Test Set through the HP-IB bus.

For the CONFIGURE, PRINT CONFIGURE, TESTS (Execution Conditions), TESTS (Printer Setup) and I/O CONFIGURE screens, [table 8-13](#) lists the fields which are restored/initialized when the \*RST command is received. The restored state or initialized value is listed below the field name. Fields which are not listed are maintained at their current value, whatever that may happen to be. All fields in the TESTS (Main Menu) screen and the TESTS (External Devices) screen are maintained at their current state/value. The current state/value of the maintained fields can be ascertained programmatically.

**Table 8-13** Screen Fields Restored/Initialized During \*RST Reset (Continued)

CONFIGURE Screen Fields	PRINT CONFIGURE Screen Fields	TESTS (Execution Conditions) Screen Fields	TESTS (Printer Setup) Screen Fields	I/O CONFIGURE
RX/TX Cntl  Auto/PTT	Print Title field is cleared	Test output location:  Crt	Test output location:  Crt	Save/Recall  Internal
RF Offset  Off		Results output:  All	Results output:  All	
(Gen)-(Anl) 0.000000		If Unit Under Test Fails:  Continue		

<b>CONFIGURE Screen Fields</b>	<b>PRINT CONFIGURE Screen Fields</b>	<b>TESTS (Execution Conditions) Screen Fields</b>	<b>TESTS (Printer Setup) Screen Fields</b>	<b>I/O CONFIGURE</b>
Range Hold Auto All		Test Procedure run mode: Continuous		
Notch Coupl None				
RF Display Freq				
RF Chan Std MS AMPS				
User Def Base Freq 800.000000				
Chan Space 30.0000				
(Gen)-(Anl) 45.000000				
RF Level Offset Off				
RF In/Out 0.0				
Duplex Out 0.0				
Antenna In 0.0				

The \*RST Reset condition in the Test Set was specifically designed to configure the instruments for manual testing of an FM radio. The \*RST Reset default display screen is the RX TEST screen. Other operational characteristics are also affected by the \*RST reset as follows:

- All pending operations are aborted.
- Measurement triggering is set to TRIG:MODE:SETT FULL;RETR REP.
- Any previously received Operation Complete command (\*OPC) is cleared.
- Any previously received Operation Complete query command (\*OPC?) is cleared.
- The Test Set's display screen is in the UNLOCKED state.
- The Power-up self-test diagnostics are not performed.
- The Contents of the SAVE/RECALL registers are not affected.
- Calibration data is not affected.
- The HP-IB interface is not reset (any pending Service Request is not cleared).
- All Enable registers are unaffected: Service Request, Standard Event, Communicate, Hardware #1, Hardware #2, Operation, Calibration, and Questionable Data/Signal.
- All Negative Transition Filter registers are unaffected: Communicate, Hardware #1, Hardware #2, Operation, Calibration, and Questionable Data/Signal.
- All Positive Transition Filter registers are unaffected: Communicate, Hardware #1, Hardware #2, Operation, Calibration, and Questionable Data/Signal.
- The contents of the RAM memory are unaffected.
- The contents of the Output Queue are unaffected.
- The contents of the Error Queue are unaffected.

## Device Clear (DCL) HP-IB Bus Command

The Device Clear (DCL) Reset is accomplished by sending the DCL message to the Test Set through the HP-IB bus.

The DCL command clears the Input Buffer and Output Queue, clears any commands in process, puts the Test Set into the Operation Complete idle state, and prepares the Test Set to receive new commands. The DCL bus command does not change any settings or stored data (except as noted previously), interrupt front panel I/O, interrupt any Test Set operation in progress (except as noted



previously), or change the contents of the Status Byte Register (other than clearing the MAV bit as a consequence of clearing the Output Queue).

The DCL bus command has no effect on the I/O CONFIGURE, CONFIGURE, PRINT CONFIGURE, or TESTS (Main Menu, Execution Conditions, External Devices, Printer Setup) screens.

Other operational characteristics are also affected by the DCL bus command as follows:

- The Power-up self-test diagnostics are not performed.
- The HP-IB interface is not reset (any pending Service Request is not cleared)
- Measurement triggering is not affected.
- Calibration data is not affected.
- The Contents of the SAVE/RECALL registers are not affected.
- All Enable registers are unaffected: Service Request, Standard Event, Hardware #1, Hardware #2, Operation, Calibration, and Questionable Data/Signal.
- All Negative Transition Filter registers are unaffected: Hardware #1, Hardware #2, Operation, Calibration, and Questionable Data/Signal.
- All Positive Transition Filter registers are unaffected: Hardware #1, Hardware #2, Operation, Calibration, and Questionable Data/Signal.
- The contents of the RAM memory are unaffected.
- The contents of the Error Queue are unaffected.
- The state of the Test Set's display screen is unaffected.

## **Selected Device Clear (SDC) HP-IB Bus Command**

The Selected Device Clear (SDC) Reset is accomplished by sending the SDC message to the Test Set through HP-IB. The Test Set responds to the Selected Device Clear (SDC) and the Device Clear (DCL) bus commands equally. Refer to the "[Device Clear \(DCL\) HP-IB Bus Command](#)" on page 334 for a description of the effects of the SDC Reset.

## Interface Clear (IFC) HP-IB Bus Command

The Interface Clear (IFC) Reset is accomplished by having the Active Controller send the ABORT message to the HP-IB bus (ABORT message = IFC bus control line TRUE for 100 ms).

The IFC bus command unconditionally terminates all HP-IB bus activity and the Test Set is unaddressed.

The IFC bus command has no effect on the I/O CONFIGURE, CONFIGURE, PRINT CONFIGURE, or TESTS (Main Menu, Execution Conditions, External Devices, Printer Setup) screens.

Other operational characteristics are also affected by the IFC bus command as follows:

- The Power-up self-test diagnostics are not performed.
- The HP-IB interface is not reset (any pending Service Request is not cleared).
- The Contents of the SAVE/RECALL registers are not affected.
- Measurement triggering is not affected.
- Calibration data is not affected .
- All Enable registers are unaffected: Service Request, Standard Event, Hardware #1, Hardware #2, Operation, Calibration, and Questionable Data/Signal.
- All Negative Transition Filter registers are unaffected: Hardware #1, Hardware #2, Operation, Calibration, and Questionable Data/Signal.
- All Positive Transition Filter registers are unaffected: Hardware #1, Hardware #2, Operation, Calibration, and Questionable Data/Signal.
- The contents of the RAM memory are unaffected.
- The contents of the Error Queue are unaffected.
- The contents of the Output Queue are unaffected.
- The state of the Test Set display screen is not affected

---

## Passing Control

Communications on the HP-IB bus are accomplished according to a precisely defined set of rules (IEEE 488.1 and 488.2 Standards). Communication (data transfer) is accomplished by designating one device to be a talker (source of data or commands) and designating one or more devices to be listeners (receivers of data or commands). The device on the bus responsible for designating talkers and listeners is the Controller.

The structure of the HP-IB bus allows for more than one Controller to be connected to the bus at the same time. As a means of ensuring that orderly communications can be established on power-up or when communications have failed, the rules state that only one Controller can unconditionally demand control of the bus (through the IFC line). This controller is referred to as the System Controller. There can be only one System Controller connected to the bus at any time.

As a means of ensuring orderly communications in environments where more than one controller is connected to the bus, the rules state that only one Controller can be actively addressing talkers and listeners at any given time. This device is referred to as the Active Controller. The System Controller is the default Active Controller on power-up or after a bus reset. Controllers which are not the Active Controller are referred to as Non-Active Controllers. The Active Controller can pass control of device addressing to one of the Non-Active Controllers. Additionally, Non-Active Controllers can request control from the Active Controller.

The process by which the Active Controller passes device addressing responsibility to a Non-Active Controller is referred to as Passing Control. The Active Controller must first address the prospective new Active Controller to Talk, after which it sends the Take Control Talker (TCT) message across the bus. If the other Controller accepts the message it assumes the role of Active Controller and the previous Active Controller becomes a Non-Active Controller.

The Test Set has bus control capability (Active/Non-Active Controller). Additionally the Test Set can be also be configured as the System Controller. By definition then, the Test Set has the capability to demand control, pass control, accept control, and request control of the bus depending upon its configuration, its current operating mode, and the system configuration. Many possibilities for passing control among several controllers on the same bus exist. Attempting to identify all the possible techniques of passing control in such a system is beyond the scope of this document (refer to the IEEE 488.1 and 488.2 Standards for additional information).

## Configuring the Test Set as the System Controller

To configure the Test Set as a System Controller, select the I/O CONFIGURE screen, position the cursor to the Mode field using the Cursor Control knob, highlight the Mode field by pushing the Rotary Knob, select Control from the Choices menu. As a consequence of setting the Test Set to be the System Controller it will also become the Active Controller. The letter C appears in the upper-right corner of the display to indicate that the Test Set is now the Active Controller.

If the Test Set is the only controller on the bus it must be configured as the System Controller. If the Test Set is not the only controller on the bus, then whether or not it is configured as the System Controller would depend upon three issues:

1. whether or not other controllers have System Controller capability
2. which controller will be the Active Controller upon power-up
3. which Controller will be monitoring the bus to determine if communications have failed (only the System Controller can unconditionally demand control of the bus and reset it to a known state using the IFC line)

Ensure that only one Controller connected to the bus is configured as the System Controller or bus conflicts will occur.

## When Active Controller Capability is Required

The Test Set must be the Active Controller on the bus under the following conditions:

1. whenever the Test Set needs to control any device connected to the HP-IB bus, such as an external disk drive, an external printer, or an external instrument
2. whenever a screen image is printed to an external HP-IB printer
3. Whenever an instrument configuration is saved or recalled from an external HP-IB disk drive
4. Whenever running any HP 11807 Radio Test Software package which uses an external HP-IB device such as a disk drive, a printer, or an instrument
5. Whenever running any IBASIC program which uses an external HP-IB device such as a disk drive, a printer, or an instrument

## Passing Control to the Test Set

Control is passed to the Test Set when it is addressed to TALK and then receives the Take Control Talker (TCT) command. The programming or controller command which implements the pass control protocol as

outlined in the IEEE 488.1 and 488.2 Standards is language/controller specific. Refer to the appropriate language/controller documentation for specific implementations.

Before passing control to the Test Set the Active Controller should send the Test Set the address to use when passing control back. This is accomplished using the \*PCB Common Command. The \*PCB command tells the Test Set which address should be used when passing control back to another bus controller. Before passing bus control to the Test Set, the currently active controller can use the \*PCB command to tell the Test Set where to send the Take Control (TCT) command when the Test Set is ready to give up active control of the bus. The command is followed by a number which contains the bus address of the device that should become the next active controller. The number must round to an integer in the range 0 to 30 decimal. The command may be followed by two numbers. The first will be used as the primary address, the second as the secondary address of the next active controller.

## Passing Control Back to Another Controller

The Test Set has two methods of passing control back to another controller: 1) automatically and 2) using the IBASIC PASS CONTROL command from an IBASIC program. The two methods are described in the following sections.

### Passing Control Back Automatically

The Test Set will automatically pass control back to the controller whose address was specified in the \*PCB Common Command or to a default address of 0 (decimal) if no \*PCB command was received. Control will automatically be passed under the following conditions:

#### **Test Set is the Active Controller and an IBASIC Program is Running.**

- The IBASIC program running in the Test Set is PAUSED.
- The IBASIC program running in the Test Set finishes executing.
- An IBASIC RESET occurs while the IBASIC program is running.
- Control is passed back immediately if the System Controller executes a bus reset (IFC).

### **Test Set is the Active Controller and no IBASIC Program is Running.**

- Control will be passed back within 10 seconds of receiving bus control if no controller commands are executed (such as printing a screen image to an HP-IB printer).
- Control is passed back immediately if the System Controller executes a bus reset (IFC).
- Control is passed back at the completion of a controller command (such as printing a screen image to an HP-IB printer).

### **Passing Control Back Using IBASIC PASS CONTROL Command**

The Test Set will pass control back to another Controller when the IBASIC PASS CONTROL command is issued while an IBASIC program is running on the built-in IBASIC Controller. Refer to the *HP Instrument BASIC User's Handbook* for a complete description of the IBASIC PASS CONTROL command.

## **Requesting Control using IBASIC**

The Test Set has the capability to request control of the bus from the Active Controller from a running IBASIC program using the IBASIC command EXECUTE ("REQUEST\_CONTROL"). When the EXECUTE ("REQUEST\_CONTROL") command is executed from a running IBASIC program, the Request Control bit, bit 1, of the Test Set's Standard Event Status Register is set to the TRUE, logic 1, condition. The Active Controller detects the request in the Test Set's Standard Event Status Register either as a result of an SRQ indication by the Test Set or by some polling routine which periodically checks bit 1 of the Standard Event Status Register of all potential controllers on the bus. The Active Controller would then send the Test Set the address to which the Test Set is to later pass control using the \*PCB Common Command. The Active Controller would then pass control to the Test Set.

## **Pass Control Examples**

The following examples illustrate how pass control could be implemented in two of the common Test Set operating configurations:

1. Test Set controlled by an external controller, and
2. Test Set running an IBASIC program with an external Controller connected to HP-IB bus.

## Passing Control While the Test Set is Controlled by an External Controller

This example illustrates passing control between the Test Set and an external controller while the Test Set is being controlled by the external controller. In this mode the Test Set is NOT configured as the System Controller. Generally speaking, in this mode of operation the Test Set is considered just another device on the HP-IB bus and its Controller capabilities are not used. However, it may be desirable, under certain conditions, to print a Test Set screen to the HP-IB printer for documentation or program debugging purposes. With manual intervention it is possible to have the Active Controller pass control to the Test Set, have the operator select and print the desired screen, and then pass control back to the formerly Active Controller. The following steps outline a procedure for accomplishing this task. The example is based upon having an HP 9000 Series 300 Workstation as the external controller connected to the Test Set through the HP-IB bus. Further, it assumes that the HP-IB interface in the HP 9000 Controller is set to the default select code of 7 and address of 21.

1. If a program is running on the HP 9000 Workstation, PAUSE the program.
2. Put the Test Set in local mode (press the LOCAL key on the front panel).
3. Configure the Test Set to print to the HP-IB printer using the PRINT CONFIGURE screen.
4. Configure the Test Set to display the screen to be printed.
5. From the keyboard of the HP 9000 Workstation type in and execute the following command:

### Example

```
OUTPUT 714;"*PCB 21"
```

This command tells the Test Set the address of the Controller to pass control back to.

6. From the keyboard of the HP 9000 Workstation type in and execute the following command:

### Example

```
PASS CONTROL 714
```

This command passes control to the Test Set.

7. Put the Test Set in local mode (press the LOCAL key on the front panel).
8. Press SHIFT , then TESTS on the front panel of the Test Set to print the screen.

9. After the Test Set finishes printing the screen it will automatically pass control back to the HP 9000 Workstation.

### **Passing Control Between an External Controller and the Test Set with an IBASIC Program Running.**

The following example program illustrates the passing of control between an external Controller and the Test Set while an IBASIC program is running in the Test Set. The example is based upon having an HP 9000 Series 300 Workstation as the external controller connected to the Test Set through the HP-IB bus. Further, it is based on the assumption that the HP-IB interface in the HP 9000 Controller is set to the default select code of 7 and address of 21. In this example, the Test Set is NOT configured as the System Controller. This example illustrates the situation where the External Controller would perform the functions listed below.

1. Sends commands to the Test Set to cause a program to be loaded off of a Memory Card which is in the Test Set's front panel Memory Card slot.
2. Sends commands to the Test Set to run the program just loaded.
3. Passes control to the Test Set and then does other work while the Test Set is making measurements.

When the Test Set is finished making measurements and has data available for the External Controller, it passes control back to the External Controller.

4. The External Controller then requests the data from the Test Set.

The following program would run in the External Controller:

#### **Example**

```
10 COM /Hpib_names/ INTEGER Internal_hpib,Inst_address,Cntrl_state
20 COM /Cntrl_names/ Ext_cntrl_addrs,Int_cntrl_addrs
30 COM /Io_names/ INTEGER Printer_addrs,Pwr_suply_addrs
40 COM /Io_values/ REAL Meas_power,Prog_state${80},Prog_name${50}
50 COM /Reg_vals/ INTEGER Status_byte,Stdevnt_reg_val
60 !
70 Internal_hpib=7
80 Ext_cntrl_addrs=14
90 Int_cntrl_addrs=21
100 Printer_addrs=1
110 Pwr_suply_addrs=26
120 Inst_address=(Internal_hpib*100)+Ext_cntrl_addrs
130 Prog_name$="PASCTLEX:INTERNAL,4"
140 !
150 PRINTER IS CRT
160 !
170 ! Set the Controller up to respond to an SRQ from Test Set
180 ! The interrupt is generated by the Request Control bit in the
Test Set
190 ON INTR Internal_hpib CALL Pass_control
200 ENABLE INTR Internal_hpib;2
210 !
220 ! Bring Test Set to known state.
230 OUTPUT Inst_address;"*RST"
240 !
250 ! Set the Test Set to cause SRQ interrupt on Request Control
```



```

260 OUTPUT Inst_address;"*CLS"
270 OUTPUT Inst_address;"*ESE 2"
280 OUTPUT Inst_address;"*SRE 32"
290 !
300 ! Load the desired program into the Test Set from Memory Card
305 OUTPUT Inst_address;"DISP TIB" ! Display the IBASIC screen
310 OUTPUT Inst_address;"PROG:EXEC 'DISP ""&"Loading program."&""""'"
320 OUTPUT Inst_address;"PROG:EXEC 'GET ""&Prog_name$&""""'"
330 OUTPUT Inst_address;"PROG:EXEC 'DISP ""&"&""""'"
340 !
350 ! Run the program in the Test Set
360 OUTPUT Inst_address;"PROG:EXEC 'RUN'"
370 !
380 REPEAT
390     STATUS Internal_hpib,3;Cntrl_state
400     DISP "WAITING TO PASS CONTROL TO THE Test Set."
410 UNTIL NOT BIT(Cntrl_state,6)
420 !
430 REPEAT
440     STATUS Internal_hpib,3;Cntrl_state
450     DISP "WAITING FOR CONTROL BACK FROM THE Test Set"
460 UNTIL BIT(Cntrl_state,6)
470 !
480 ! Data is ready in the Test Set
490 OUTPUT Inst_address;"PROG:NUMB? Meas_power"
500 ENTER Inst_address;Meas_power
510 PRINT "Measured power = ";Meas_power
520 !
530 DISP "Program finished."
540 END
550 !
560 SUB Pass_control
570 !
580     COM /Hpib_names/ INTEGER Internal_hpib,Inst_address,Cntrl_state
590     COM /Cntrl_names/ Ext_cntrl_addrs,Int_cntrl_addrs
600     COM /Io_names/ INTEGER Printer_addrs,Pwr_suply_addrs
610     COM /Io_values/ REAL Meas_power,Prog_state${80},Prog_name${50}
620     COM /Reg_vals/ INTEGER Status_byte,Stdevnt_reg_val
630 !
640     OFF INTR Internal_hpib
650     Status_byte=SPOLL(Inst_address)
660     IF NOT BIT(Status_byte,5) THEN
670         PRINT "SRQ for unknown reason. Status Byte = ";Status_byte
680         STOP
690     END IF
700 !
710 ! Tell Test Set where to pass control back to
720 OUTPUT Inst_address;"*PCB";Int_cntrl_addrs
730 !
740 ! Put Test Set in LOCAL mode so front panel keys function
750 LOCAL Inst_address
760 !
770 PASS CONTROL Inst_address
780 !
790 ENABLE INTR Internal_hpib;2
800 !
810 SUBEND

```

The following IBASIC program would be loaded off the Memory Card and run in the Test Set:

### Example

```

10     COM /Hpib_names/ INTEGER Internal_hpib,External_hpib
20     COM /Cntrl_names/ Ext_cntrl_addrs,Int_cntrl_addrs
30     COM /Io_names/ INTEGER Printer_addrs,Pwr_suply_addrs
40     COM /Io_values/ REAL Meas_power
50     !
60     Internal_hpib=800
70     External_hpib=700
80     Ext_cntrl_addrs=21
90     Int_cntrl_addrs=14
100    Printer_addrs=1

```

```
110 Pwr_suply_addr=26
120 !
130 OUTPUT Internal_hpib;"*RST"
140 CLEAR SCREEN
150 PRINTER IS CRT
160 !
170 EXECUTE ("REQUEST_CONTROL")
180 !
190 Try_again: !
200 ON ERROR GOTO Not_active_cntrl
210 DISP "WAITING TO GET CONTROL"
220 OUTPUT External_hpib;" !If OUTPUT successful then Active
Controller
230 !If OUTPUT not successful then not Active Controller
240 DISP "TEST SET NOW ACTIVE CONTROLLER."
250 CALL Start_program
260 !
270 Pass_back: !
280 DISP "PASSING CONTROL BACK"
290 !Control is passed back automatically when the program stops
300 !Control is passed back to address specified by *PCB command
310 DISP "PROGRAM FINISHED"
320 STOP
330 !
340 Not_active_cntrl: !
350 OFF ERROR
360 DISP "CHECKING FOR ERROR"
370 IF ERRN=173 THEN
380 GOTO Try_again
390 ELSE
400 PRINT "ERROR =" ;ERRN
410 STOP
420 END IF
430 !
440 END
450 !
460 SUB Start_program
470 !
480 COM /Hpib_names/ INTEGER Internal_hpib,External_hpib
490 COM /Cntrl_names/ Ext_cntrl_addr,Int_cntrl_addr
500 COM /Io_names/ INTEGER Printer_addr,Pwr_suply_addr
510 COM /Io_values/ REAL Meas_power
520 !
530 PRINT "SETTING POWER SUPPLY"
540 OUTPUT External_hpib+Pwr_suply_addr;"IMAX 8;ISET 5"
550 OUTPUT External_hpib+Pwr_suply_addr;"VMAX 15;VSET 13.2"
560 !
570 PRINT "SETTING UP INTERNAL INSTRUMENTS"
580 OUTPUT Internal_hpib;"RFG:FREQ 850.030 MHz;AMPL -40 dBm"
590 OUTPUT Internal_hpib;"AFG1:FREQ 3 KHZ;DEST 'FM';FM 3 KHZ"
600 OUTPUT Internal_hpib;"DISP TX;MEAS:RFR:POW?"
610 ENTER Internal_hpib;Meas_power
620 !
630 OUTPUT External_hpib+Printer_addr;"Measured power =
";Meas_power
640 !
650 OUTPUT External_hpib+Pwr_suply_addr;"VSET 0"
660 !
670 SUBEND
```

---

## 9

# Memory Cards/Mass Storage

This chapter contains information about using the mass storage devices available in the Test Set for storing and retrieving program and data files. Access to the mass storage devices in the Test Set was designed primarily for the built-in IBASIC Controller. The Test Set's mass storage devices are not directly accessible by an external controller. The programming examples used in this chapter apply only to the Test Set's built-in IBASIC Controller.

---

### NOTE

Indirect access to the Test Set's mass storage devices is available through the PROGRAM:EXECute command. Refer to the *Standard Commands for Programmable Instruments (SCPI)* for generic information on the PROGRAM:EXECute command.

The IBASIC programming examples are provided to assist the programmer in understanding the use of the Test Set's mass storage devices. They are not intended to be a comprehensive description of the IBASIC mass storage commands and procedures. For detailed information on IBASIC commands, refer to the *HP Instrument BASIC User's Handbook*.

---

## Default File System

The Test Set's default file system is the Logical Interchange Format (LIF) System. The LIF file system is used by Hewlett-Packard BASIC on the HP 9000 Series 200/300 Workstations. See "[LIF File Naming Conventions](#)" on [page 355](#) for further information on the LIF file system. The Test Set's file system supports both LIF and DOS. The media format (DOS or LIF) is determined automatically by the Test Set's file system when the mass storage device is first accessed and the appropriate format is used from then on for mass storage operations.

**Table 9-1 Test Set Default File System**

Activity	Default File System
Manual front-panel operations SAVE/RECALL register access TESTS Subsystem file access Signaling Decoder NMT file access	LIF
IBASIC mass storage operations LIF is default, DOS is also supported	LIF
HP-IB commands for <ol style="list-style-type: none"> <li>1. SAVE/RECALL register access</li> <li>2. TESTS Subsystem file access</li> <li>3. Signaling Decoder NMT file access</li> </ol>	LIF
TESTS Subsystem <ol style="list-style-type: none"> <li>1. Procedure files</li> <li>2. Library files</li> <li>3. Code files</li> </ol>	LIF

## Mass Storage Device Overview

As shown in [figure 9-1 on page 348](#), the Test Set has both internal and external mass storage devices. There are five types of mass storage devices in the Test Set:

1. On-board random-access memory disk (RAM disk) located on the Test Set's internal memory board
2. On-board read-only memory disk (ROM disk) located on the Test Set's internal memory board
3. Internal static random access memory (SRAM) cards which are inserted into the Test Set's front-panel Memory Card slot
4. Internal read-only memory (ROM) cards (also called One-Time Programmable or OTP cards) which are inserted into the Test Set's front-panel Memory Card slot

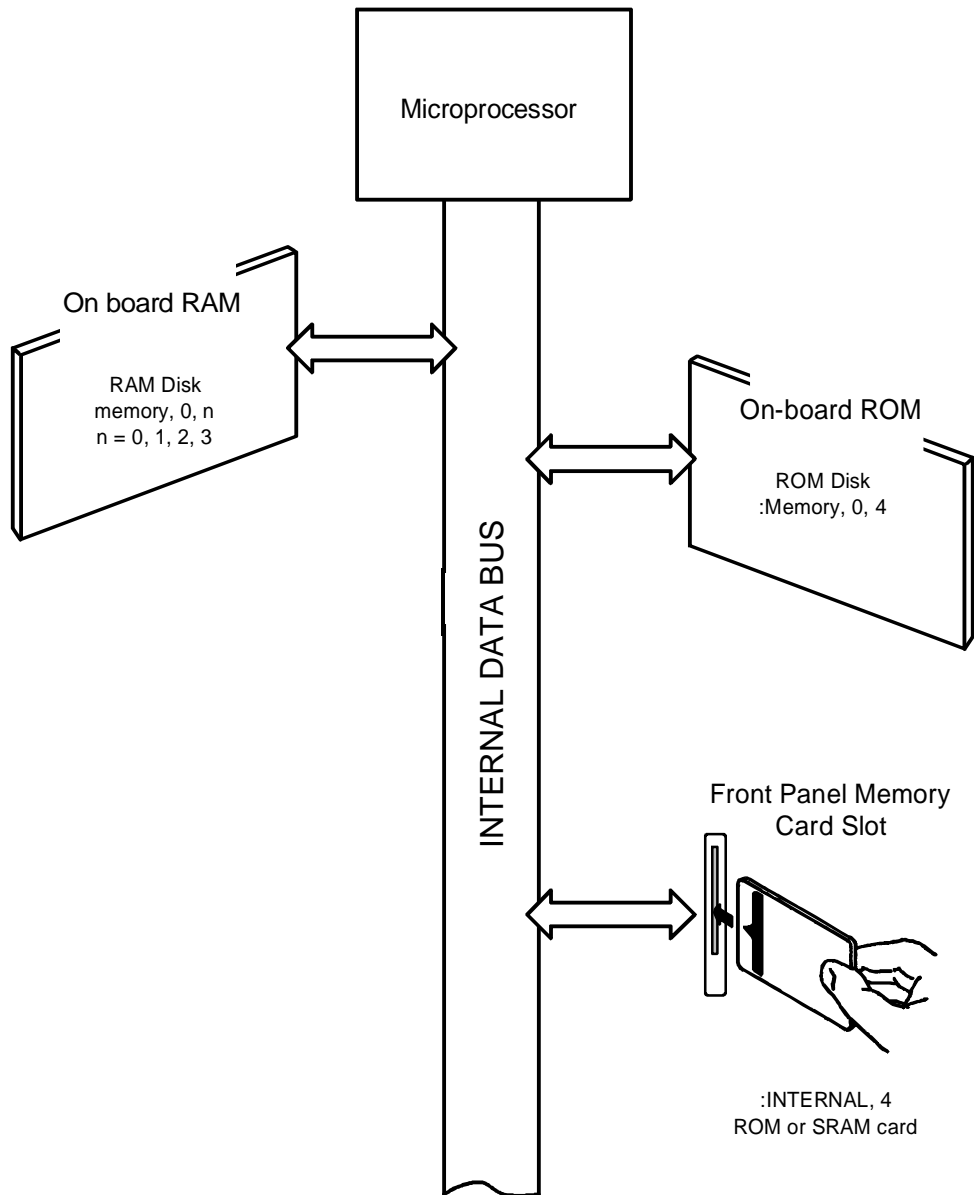
---

### NOTE

The hardware for reading-from and writing-to memory cards is located internal to the Test Set. Therefore, the static random access memory (SRAM) cards and the read only memory (ROM) cards are considered internal to the Test Set even though the physical media must be inserted into the Test Set's front-panel Memory Card slot.

---

**Figure 9-1** Internal and External Mass Storage Devices



ch5drw1.drw

Programs and data can be retrieved from any of these mass storage devices. Programs and data can only be stored to RAM disk, external disk, or SRAM card mass storage devices. The IBASIC file system supports both the LIF (Hewlett-Packard's Logical Interchange Format) file system and the MS-DOS (Microsoft Disk Operating System) file system.

The following paragraphs provide an overview of the five types of mass storage devices.

RAM Disk Mass Storage Overview

<b>Mass Storage Name</b>	<b>Mass Storage Type</b>	<b>Physical Location</b>	<b>Mass Storage Volume Specifier</b>	<b>Media Type</b>	<b>Supported File System(s)</b>
RAM Disk	Non-volatile random access memory	Test Set's internal memory board	":MEMORY,0,unit number" unit number = 0, 1, 2, or 3 default = 0	N/A	LIF, DOS

Typical Uses

- Temporary program and data storage
- Temporary Save/Recall register storage

Comments

- Easily overwritten or erased
- Not recommended for permanent program or data storage
- Unit 0 can be overwritten by the RAM\_MNG utility program (ROM Disk)
- Unit 1 can be overwritten by the COPY\_PL utility program (ROM Disk)
- Units 2 and 3 are not overwritten by any ROM Disk utility program

**Table 9-2** ROM Disk Mass Storage Overview

<b>Mass Storage Name</b>	<b>Mass Storage Type</b>	<b>Physical Location</b>	<b>Mass Storage Volume Specifier</b>	<b>Media Type</b>	<b>Supported File System(s)</b>
ROM Disk	Read-only memory	Test Set internal memory board	":MEMORY,0,4"	N/A	LIF

Typical Uses

- Permanent storage of factory supplied utility programs
- Permanent storage of factory supplied diagnostic programs

Comments

- Non-erasable
- Not available for user program or data storage
- Not available for Save/Recall register storage

**Table 9-3** SRAM Card Mass Storage Overview

<b>Mass Storage Name</b>	<b>Mass Storage Type</b>	<b>Physical Location</b>	<b>Mass Storage Volume Specifier</b>	<b>Media Type</b>	<b>Supported File System(s)</b>
SRAM Memory Card	Static Random-Access Memory Card	Plugs into Memory Card slot on front panel of Test Set	":INTERNAL,4"	EPSON SRAM Memory Card	LIF, DOS

Typical Uses

- Semi-permanent program and data storage
- Semi-permanent Save/Recall register storage



Comments

- Low capacity
- Contents retained by on-card lithium battery
- Contents lost if on-card battery removed while card not in Test Set Memory Card slot
- Recommended as primary mass storage device for program and data storage

ROM Card Mass Storage Overview

<b>Mass Storage Name</b>	<b>Mass Storage Type</b>	<b>Physical Location</b>	<b>Mass Storage Volume Specifier</b>	<b>Media Type</b>	<b>Supported File System(s)</b>
ROM or OTP Memory Card	Read-only Memory Card	Plugs into Memory Card slot on front panel of Test Set	":INTERNAL,4"	EPSON ROM Memory Card	LIF

Typical Uses

- Permanent storage of factory supplied application programs
- Permanent storage of factory supplied utility programs
- Permanent storage of factory supplied diagnostic programs

Comments

- Non-erasable
- Not available for user program or data storage
- Not available for Save/Recall register storage

## Default Mass Storage Locations

### Built-in IBASIC Controller

The default mass storage location for the built-in IBASIC Controller is the front panel memory card slot (mass storage volume specifier ":INTERNAL,4") after any of the following conditions:

power-up

initializing RAM with the SERVICE screen's RAM Initialize function

The mass storage location for the built-in IBASIC Controller can be changed using the MASS STORAGE IS command. Refer to the *HP Instrument BASIC Users Handbook* for further information on the MASS STORAGE IS command.

### Save/Recall Registers

The default mass storage location for the Save/Recall registers is the Test Set's internal RAM (no mass storage volume specifier) after any of the following conditions:

- power-up
- initializing RAM with the SERVICE screen's RAM Initialize function
- resetting the Test Set using the front-panel PRESET key
- resetting the Test Set using the \*RST HP-IB Common Command

The mass storage location for Save/Recall registers can be changed using the Save/Recall field in the I/O CONFIGURE screen. The default mass storage volume specifiers for the Save/Recall register mass storage locations are as follows:

- Internal selection - (no mass storage volume specifier, registers are saved to allocated RAM space)
- Card selection (not changeable) - ":INTERNAL,4"
- RAM selection (not changeable) - ":MEMORY,0,0"
- Disk selection - the External Disk Specification field in the TESTS (External Devices) screen.

## TESTS Subsystem

The default mass storage location for the TESTS Subsystem is set using the `Select Procedure Location:` field on the TESTS (Main Menu) screen. The default mass storage volume specifiers for the TESTS Subsystem mass storage locations are as follows:

- Card selection (not changeable) - `":INTERNAL,4"`
- ROM selection (not changeable) - `":MEMORY,0,4"`
- RAM selection (not changeable) - `":MEMORY,0,0"`

## Selecting the Mass Storage Location

The IBASIC mass storage location is selected using the IBASIC Mass Storage Is command. The mass storage volume specifier for the desired mass storage location is appended to the Mass Storage Is command. Refer to the *HP Instrument BASIC User's Handbook* for further information regarding the Mass Storage Is command.

For example, to change the default mass storage location to RAM Disk unit 2, execute the following command:

```
Mass Storage Is ":MEMORY,0,2"
```

The Mass Storage Is command is keyboard and program executable; however, any changes made are lost when the Test Set is turned off or when the SERVICE screen's `RAM Initialize` function is executed.

## Mass Storage Access

Program and data files stored on the Test Set's various mass storage locations can be selectively accessed from the following screens:

- The TESTS (IBASIC Controller) screen.  
Any type of file can be accessed from this screen, either through an IBASIC program or the IBASIC command line.
- The TESTS (Main Menu) screen using the `Select Procedure Location:` and `Select Procedure Filename:` fields.  
Only *procedure* files shipped with HP 11807 software or procedure files created using the TESTS (Save/Delete Procedure) screen of the TESTS Subsystem can be accessed using these fields. When created, procedure file names are prefixed with a lower case *p* (pFM\_TEST).  
A corresponding *code* file - prefixed with a lower case *c* (cFM\_TEST) on the - must reside on the same media for the procedure to work. Refer to the TESTS screen description in the User's Guide for further information on the TESTS Subsystem.
- The TESTS (Save/Delete Procedure) screen using the `Select Procedure Location:` and `Enter Procedure Filename:` fields.  
This screen is used to create "procedure" files. When created, procedure file names are prefixed with a lower case *p* (pFM\_TEST).

Save/Recall register files, stored on the Test Set's various mass storage locations, can be accessed using the front-panel SAVE and RECALL keys.

## DOS and LIF File System Considerations

Program and data files can be stored and retrieved from IBASIC using either the DOS or LIF file system. The media format (DOS or LIF) is determined automatically by the IBASIC file system when the mass storage device is first accessed, and the appropriate format is used from then on. DOS and LIF use different file naming conventions. In addition, the Test Set uses certain file naming conventions which are unique to the Test Set. Unexpected file operation can occur if proper consideration is not given to the file naming conventions.

### File Naming Conventions

#### LIF File Naming Conventions

The LIF file system is used by Hewlett-Packard BASIC on the HP 9000 Series 200/300 Workstations. It is a flat file system, which means that it has no subdirectories. The LIF file system allows up to 10-character file names which are case sensitive. The LIF file system preserves the use of uppercase and lowercase characters for file storage and retrieval. For example, the file names File1, FILE1, file1 and FiLe1 could represent different files. LIF files cannot start with a space, and any file name longer than 10 characters is considered an error.

The Test Set's file system does not support the HFS (hierarchical file system) used with HP BASIC. Therefore, no directory path information can be used during mass storage operations with LIF files.

#### DOS File Naming Conventions

The DOS file system is used on IBM compatible personal computers. The DOS file system is hierarchical, which means it supports subdirectories. The DOS file system allows up to 8-character file names with an optional extension of up to 3 characters. The file name is separated from the extension (if it exists) with a period (.). DOS file names are case independent. The characters are stored as upper case ASCII in the DOS directory but the files may be referenced without regard to case. The DOS file system always converts any lowercase characters to uppercase when files are stored. For example, the file names File1 , FILE1 , file1 and FiLe1 all represent the single DOS file FILE1.

The period (.) may appear in the name but only to separate the file name from the extension. The period is not considered part of the file name itself. If the name portion of a DOS file name is longer than 8 characters, it is truncated to 8 characters and no error is generated. Similarly, if the extension is longer than 3 characters, it is truncated to 3 characters and no error is given.

## Test Set File Naming Conventions

The Test Set's TESTS Subsystem uses the following file naming conventions:

- The *c* prefix is used to indicate a code file and is automatically prefixed onto the file name when the program code file is stored for use by the TESTS subsystem.
- The *p* prefix is used to indicate a procedure file and is prefixed onto the file name when the file is stored by the TESTS Subsystem
- The *l* prefix is used to indicate a library file and is prefixed onto the file name when the file is created by the Program Development System for use with the TESTS Subsystem

The Test Set's Save/Recall register subsystem uses the following file naming convention:

The *\_* prefix is used to indicate a stored Save/Recall register file and is prefixed onto the file name when the file is created.

The Test Set's Signaling Decoder in NMT mode uses the following file naming convention:

- The *n* prefix is used to indicate a stored NMT file and is prefixed onto the file name when the file is created.

## Test Set File Entry Field Width

The TESTS Subsystem and the Save/Recall register subsystem have fields into which the operator enters a file name. These fields are used by the operator to enter the name of a file to be stored or loaded. The files accessed by these fields have a one-character prefix of *c*, *p*, *l*, or *\_*. The width of these fields is 9 characters. The prefix character + 9 characters = 10 characters, which conforms to the LIF file system's naming convention. Consequently these fields will hold a file name which is longer than the 8 characters allowed by the DOS file system.

## Potential File Name Conflicts

Unexpected file operation can occur if proper consideration is not given to the different file system naming conventions and the Test Set file entry field width.

- A full DOS file name is 12 characters (8 character file name + . + 3 character extension). A full DOS file name will not fit in the Test Set's file entry field.
- Trying to store a file to a LIF formatted media with a DOS file name that contains an extension will generate ERROR 53 Improper file name.
- On a DOS formatted disk, any file beginning with the letter c (upper or lower case) is considered a TESTS Subsystem code file. On a LIF formatted disk any file beginning with a lower case c is considered a TESTS Subsystem code file. If the TESTS Subsystem attempts to retrieve a file which is not a code file, the following error will be generated: Error reading code file. Check file and media.
- On a DOS formatted disk, any file beginning with the letter p (upper or lower case) is considered a TESTS Subsystem procedure file. On a LIF formatted disk, any file beginning with a lower case p is considered a TESTS Subsystem procedure file. If the TESTS Subsystem attempts to retrieve a file which is not a procedure file, the following error will be generated: Error reading procedure file. Check file and media.
- On a DOS formatted disk, any file beginning with the letter l (upper or lower case) is considered a TESTS Subsystem library file. On a LIF formatted disk, any file beginning with a lower case l is considered a TESTS Subsystem library file. If the TESTS Subsystem attempts to retrieve a file which is not a library file, the following error will be generated: Error reading library file. Check file and media.
- When reading files from mass storage to either the TESTS Subsystem (procedure, code, or library files) or the Save/Recall register Subsystem, the Test Set interprets the "." (period) as a delimiter and ignores any following characters. If TESTS Subsystem or Save/Recall register subsystem files are stored to a DOS formatted media using file extensions, the extensions will be stripped off by the Test Set before displaying the file in the file list.
- When reading files from mass storage to either the TESTS Subsystem (procedure, code, or library files) or the Save/Recall register subsystem, the Test Set strips the prefix character (c, p, l, \_) off the file name before displaying the file in the file list.

- When storing files to mass storage from either the TESTS Subsystem (procedure, code, or library files) or the Save/Recall register subsystem, the Test Set puts the prefix character (c, p, l, \_) onto the file name, making the file name 1 character longer than that displayed in the file name entry field. If the file is being stored to a DOS formatted media (8-character file name) and the file name specified in the file name entry field is 8 characters (ABCDEFGH) the last character will be silently truncated when the file is stored (PABCDEFG).
- When copying LIF named files to a DOS formatted media, the file name is silently truncated to 8 characters since DOS only allows 8-character file names. This could result in `ERROR 54 Duplicate File Name`.
- When storing or deleting files to a DOS formatted media, the file name is silently truncated to 8 characters since DOS only allows 8-character file names. This could result in `ERROR 54 Duplicate File Name`.

## File Naming Recommendations

If switching between media types (DOS and LIF) or operating exclusively in DOS the following naming conventions are recommended.

- Ensure that only TESTS Subsystem procedure files begin with the letter p (upper or lower case).
- Ensure that only TESTS Subsystem library files begin with the letter l (upper or lower case).
- Ensure that only TESTS Subsystem code files begin with the letter c (upper or lower case).
- Ensure that only user-written NMT test files begin with the letter n (upper or lower case).
- Avoid using DOS file extensions.
- If possible, only use file names of 7 characters or less for Save/Recall registers or TESTS Subsystem files (prefix character + 7 characters = 8-character DOS file name limit). This will avoid silent truncation of file names which leads to many of the problems discussed under "[Potential File Name Conflicts](#)" on page 357.

## Initializing Media for DOS or LIF File System

The INITIALIZE command is used to initialize a media (external hard disk, external 3.5-inch floppy disk, Epson SRAM Card, PCMCIA SRAM Card and RAM Disk) for use with the DOS or LIF file system. The DOS or LIF file system is specified with the parameter. LIF is the default.



## Test Set File Types

The Test Set file system supports the following file types:

### Storing Code Files

Two IBASIC commands are available for storing program code to a mass storage location: SAVE and STORE. The type of file created by the Test Set's file system when the program code is stored, is dependent upon the format of the media being used. The type of file created verses the media format is outlined in [table 9-4](#).

**Table 9-4**      **Stored Program Code File Types**

	<b>DOS Formatte d Media</b>	<b>LIF Formatted Media</b>
SAVE	DOS	ASCII
STORE	DOS	HP-UX

Files that have been stored using the SAVE command must be retrieved using the GET command:

**Example**

```
SAVE "FM_TEST:;,704,1"
```

```
GET "FM_TEST:;,704,1"
```

Files that have been stored using the STORE command must be retrieved using the LOAD command:

**Example**

```
STORE "FM_TEST:;,704,1"
```

```
LOAD "FM_TESTS:;,704,1"
```

## TESTS Subsystem DOS File Restrictions

The Test Set uses IBASIC revision 1.0. The IBASIC 1.0 file system cannot distinguish between DOS files that have been “SAVED” and those that were “STORED.” As shown in [table 9-4 on page 359](#), SAVE and STORE both produce a file type DOS. This can result in undesired operation when trying to run a Test procedure from the TESTS (Main Menu) screen.

The process for running a Test Procedure is described below. The potential problem is described in step 3.

1. The procedure file location is selected using the `Select Procedure Location:` field.
2. The desired procedure file is selected using the `Select Procedure Filename:` field. When the procedure file is selected, the Test Set loads the specified procedure file into memory. One of the pieces of information in the procedure file is the name of the code file used with that procedure.
3. The `Run Test` softkey is selected. When the `Run Test` softkey is selected the Test Set attempts to load the code file into memory. If the code file is located on a DOS formatted media the Test Set will attempt to GET the file (the Test Set assumes the file was stored using the SAVE command). If the code file was stored to the DOS formatted media using the STORE command an `ERROR 58 Improper file type` is generated.

If an `ERROR 58 Improper file type` is generated the code file must be loaded into memory and then stored back to mass storage using the SAVE command as follows:

1. Access the TESTS (IBASIC Controller) screen and LOAD the code file into the Test Set.
2. Delete the stored code file from the mass storage location using the IBASIC PURGE command.
3. SAVE the program as a Code file, using a lower-case *c* as a prefix, to the *same mass storage location* as the original code file.

The IBASIC 1.0 file system can distinguish between LIF files that have been “SAVED” and those that were “STORED.” Consequently the Test Set can determine whether to use a GET or a LOAD on a code file which is located on a LIF formatted media.

---

## Using the ROM Disk

The Test Set comes with several Test Procedures stored on the internal ROM disk. These Test procedures provide instrument diagnostic utilities, periodic calibration utilities, memory management utilities, a variety of general purpose utilities, and several IBASIC demonstration programs.

To see a brief description of what each procedure does perform the following steps:

1. Display the TESTS (Main Menu) screen by selecting the front-panel TESTS key.
2. Using the rotary knob, select the `Select Procedure Location:` field and choose ROM from the choices.
3. Using the rotary knob, select the `Select Procedure Filename` field. A list of Test Procedures stored on the ROM disk is displayed in the `Choices:` field. Using the rotary knob, select the Test Procedure of interest.
4. A brief description of the Test Procedure will be displayed in the `Description` field.

ROM DISK cannot be written to for user storage.

The ROM Disk's mass storage volume specifier is `":MEMORY,0,4"`

For example: to catalogue the contents of the ROM Disk from the TESTS (IBASIC Controller) screen enter:

**Example**

```
30 OUTPUT 814;"AFAN:DEMP:GAIN 20 dB"
```

---

## Using Memory Cards

OTP (One Time Programmable) cards provide removable read-only storage. File editing and erasure are not possible. These cards cannot be programmed by the Test Set; they require a special memory card programmer to save files.

SRAM cards provide removable read/write memory for your files, similar to a flexible disk. Data can be stored, re-stored, read, or erased as needed.

SRAM memory cards require a battery to maintain stored information.

**Table 9-5**      **Memory Card Part Numbers**

Memory	Type	Part Number
32 kilobytes	SRAM	HP 85700A
128 kilobytes	OTP	HP 85701A
128 kilobytes	SRAM	HP 85702A
256 kilobytes	OTP	HP 85703A
256 kilobytes	SRAM	HP 85704A
512 kilobytes	SRAM	HP 85705A
512 kilobytes	OTP	HP 85706A

## Inserting and Removing Memory Cards

**Figure 9-2** illustrates how to insert a memory card into the Test Set's front panel. To remove a memory card, simply pull it out.

The Test Set's memory-card label is marked with an arrow that must be inserted on the same side as the arrow shown on the front-panel slot.

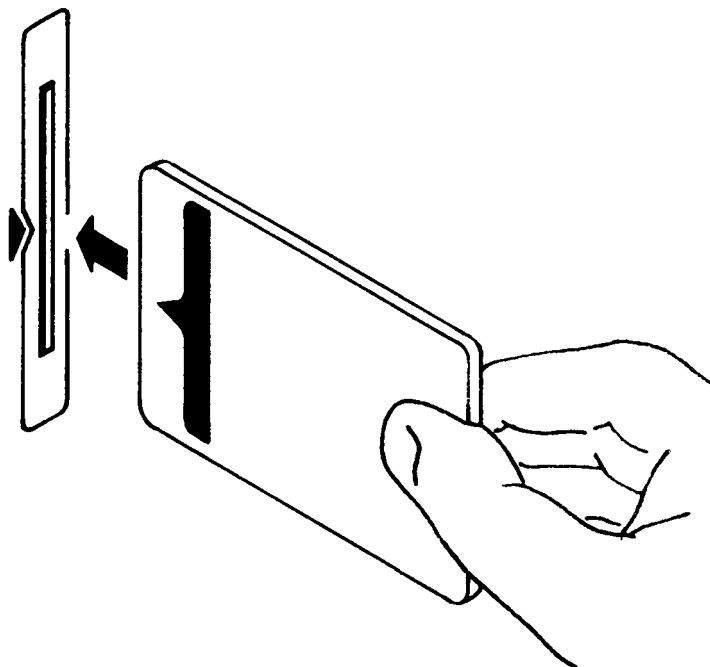
---

**NOTE**

Memory cards may be inserted and removed with the Test Set powered on or off.

---

**Figure 9-2**      **Inserting a Memory Card**

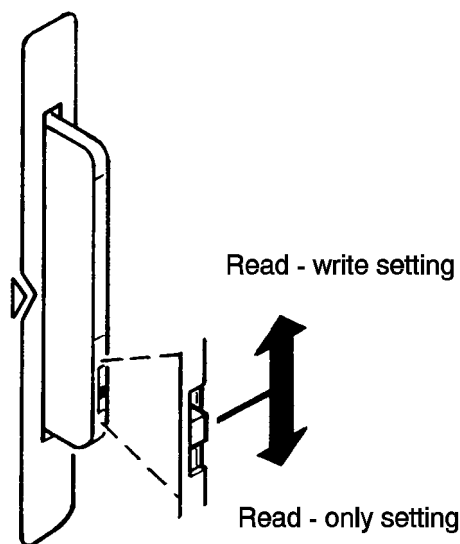


## Setting the Write-Protect Switch

The SRAM memory card's write-protect switch lets the user secure its contents from being overwritten or erased. The switch has two positions (see [figure 9-3](#)):

- *Read-write* – The memory-card contents can be changed or erased, and new files may be written on the card.
- *Read-only* – The memory-card contents can be read by the Test Set, but cannot be changed or erased.

**Figure 9-3**      **Setting the SRAM Write-Protect Switch**



## The Memory Card Battery

SRAM memory cards use a lithium battery to power the card. Listed below are the batteries for the Test Set's SRAM cards. SRAM cards typically retain data for over 1 year at 25° C. To retain data, the battery should be replaced annually.

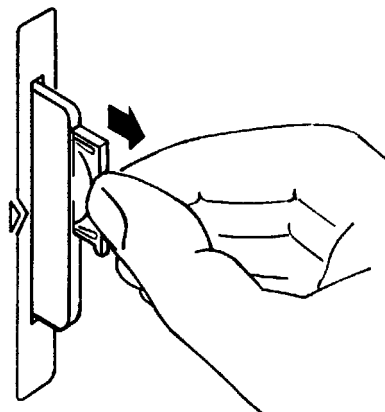
SRAM Card Battery Part Numbers - CR2016 or HP 1420-0383

### Replacing the Battery

1. Turn the Test Set on and insert the memory card. An inserted memory card takes power from the Test Set, preventing the card's contents from being lost.
2. Hold the memory card in the slot with one hand and pull the battery holder out with your other hand. (See [figure](#) .)
3. Install the battery with the side marked "+" on the same side marked "+" on the battery holder. Avoid touching the flat sides of the battery, finger oils may contaminate battery contacts in the memory-card.

4. Re-insert the battery holder into the memory card.
5. Remove the memory card from the Test Set.

#### Replacing the Memory Card's Battery



---

**WARNING**

**Do not mutilate, puncture, or dispose of batteries in fire. The batteries can burst or explode, releasing hazardous chemicals. Discard unused batteries according to the manufacturer's instructions.**

---

### Memory Card Mass Storage Volume Specifier

The front-panel memory card slot's mass storage volume specifier is ":INTERNAL,4" and is the default mass storage device for the Test Set. For example, to catalogue the contents of a memory card from the TESTS (IBASIC Controller) screen, execute the following IBASIC command:

**Example**

```
C376AT ":INTERNAL,4"
```

or, if the mass storage location has not been changed,

**Example**

```
CAT
```

If the MSI (Mass Storage Is) command has been used to change the mass storage location to a different device, the ":INTERNAL,4" designation must be used to access the memory card slot. Any changes to the mass storage location made with the MSI (Mass Storage Is) command are lost when the Test Set is turned off.

## Memory Cards and Initialization

There are several types of memory cards available, and the following cards are used with the Test Set:

- SRAM: used for Save/Recall and data storage
- Flash ROM: used when upgrading firmware
- OTP (One Time Programmable): used for HP software

Flash ROM cannot be used for collecting data and Save/Recall.

Data cannot be loaded on Flash RAM and OTP cards with the Test Set's memory card slot. SRAM can be initialized with the Test Set.

### Initializing SRAM Cards for Save/Recall and Data Collection

1. Insert the SRAM card into the slot. If the card is uninitialized, a message will appear at the top of the display.
2. Press **shift** and **IO Cnfg** to display the I/OCONFIGURE screen.
3. Using the knob, locate the **FORMAT CARD** field.
4. Select the **FORMAT CARD** field. A prompt will appear at the top of the display. Pressing **yes** will erase and initialize the card.



## Backing Up Procedure and Library Files

Making a backup copy of procedure and library files helps guard against file loss due to memory card (or battery) failure.

### Using the COPY\_PL ROM Program

The COPY\_PL procedure on the internal ROM Disk will make backup copies of TESTS Subsystem's Procedure and Library files onto a second SRAM memory card, and can also initialize an uninitialized SRAM memory card. This program does not make backup copies of TESTS Subsystem's code files, or copy any type of file to OTP memory cards.

The COPY\_PL procedure is designed for use with HP 11807 software to make backup copies of Hewlett-Packard supplied TESTS Subsystem's Procedure and Library files or user-generated TESTS Subsystem's Procedure and Library files.

To run COPY\_PL:

1. Access the TESTS (Main Menu) screen.
2. Select the Select Procedure Location: field and choose ROM.
3. Select the Select Procedure Filename: field and select COPY\_PL.
4. Select the Run Test softkey to start the procedure.
5. Follow the displayed instructions.

## Copying Files Using IBASIC Commands

Files can be copied from one mass storage device to another using the IBASIC COPY command. For example, to copy a file from a memory card to the left drive of an external dual-disk drive with a mass storage volume specifier of ":",702,0", execute the following IBASIC command from the TESTS (IBASIC Controller) command line:

### Example

```
COPY "FM_TEST:INTERNAL,4" TO "FM_TEST:,704,0"
```

“Stored” or “saved” files on one memory card can be copied to another memory card as follows:

- Insert the memory card containing the file to be copied.
- LOAD or GET<sup>1</sup> the desired file from the memory card into the Test Set.
- Remove the original memory card.
- Insert the destination memory card in the Test Set.
- STORE or SAVE\* the file to the destination memory card.

## Copying an Entire Volume

An entire volume can be copied from one mass storage device to the same type of mass storage device using the volume copy form of the COPY command. The destination volume must be as large as, or larger than, the source volume. The directory and any files on the destination volume are destroyed. The directory size on the destination volume becomes the same size as the source media. Disc-to-disc copy time is dependent on the mass storage device type. The volume copy form of the COPY command was designed to copy like-media to like-media and like-file-systems to like-file-systems. For example, to copy the entire contents of one internal RAM disk to another internal RAM disk, execute the following IBASIC command from the TESTS (IBASIC Controller) command line:

### Example

```
COPY ":MEMORY,0,0" TO ":MEMORY,0,1"
```

---

1. See ["Storing Code Files" on page 359](#) for information about the LOAD, GET, STORE, and SAVE commands.

---

NOTE

Using the volume copy form of the COPY command can produce unexpected results. For example, using the volume copy form to copy the contents of a 64 Kbyte SRAM card to an external HP-IB 630-KByte floppy disk will result in the external floppy disk having a capacity of only 64 Kbyte when the volume copy is finished. Furthermore all files on the floppy disk before the volume copy was executed will be lost and *are not recoverable*. Additionally, the file system type on the source media (LIF or DOS) is forced onto the destination media. Caution should be exercised when using the volume copy form of the COPY command.

The Test Set only supports the following types of volume copy using the volume copy form of the COPY command:

1. Like- media to like-media (RAM disk to RAM disk, external floppy to external floppy, and so forth)
2. Like-file-system to like-file-system (DOS to DOS, LIF to LIF)

All other types of volume copy are unsupported and will produce unexpected results or system errors.

---

Using wildcards in the COPY command can eliminate the need to use the volume form of the COPY command. Refer to the *HP Instrument BASIC User's Handbook* for further information on wildcards and their use in the COPY command.

## Using RAM Disk

RAM Disk is a section of the Test Set's internal RAM memory that has been set aside for use as a mass storage device. RAM Disk acts much the same as an external disk drive; that is, program and data files can be stored, re-stored, erased, and retrieved from the RAM Disk.

The RAM Disk is partitioned into four separate units: 0-3. Each unit is treated as a separate "disk." The size of each disk can be specified in 256-byte increments.

The four RAM Disk units are designated ":MEMORY,0,0" to ":MEMORY,0,3". For example, to catalog the contents of RAM Disk unit "0" from the TESTS (IBASIC Controller) screen, execute the following command:

### Example

```
CAT ":MEMORY,0,0"
```

Volume 0's contents can be viewed and loaded from the TESTS (IBASIC Controller) screen, the TESTS (Main Menu) screen, the TESTS (Save/Delete Procedure) screen and the Signaling Decoder screen in NMT mode. Volumes 1, 2, and 3 can only be accessed from the TESTS (IBASIC Controller) screen.

---

### NOTE

#### RAM Disk Erasure.

The contents of RAM Disk are easily lost. Unit 0 can be overwritten by the RAM\_MNG utility program (ROM Disk). Unit 1 can be overwritten by the COPY\_PL utility program (ROM Disk). The contents of all units are lost when the SERVICE screen's RAM Initialize function is executed. Therefore, RAM Disk should only be used for non-permanent, short-term storage of program or data files.

---

## Initializing RAM Disks

Each RAM Disk unit must be initialized before it can be used. Unit 0 can be initialized using the RAM\_MNG procedure stored on internal ROM Disk. Volumes 1, 2, and 3 must be initialized from the TESTS (IBASIC Controller) screen.

The optional “unit size” parameter in the following procedure specifies the memory area, in 256 byte blocks, set aside for each disk unit.

Follow these steps to initialize volumes 1, 2, or 3:

1. Access the TESTS (IBASIC Controller) screen.
2. Using the rotary knob or an external terminal, enter and execute the IBASIC command:

### Example

```
INITIALIZE ":MEMORY,0,<unit number 1-3>",<unit size>
```

For example:

### Example

```
INITIALIZE ":MEMORY,0,1",50
```



---

---

# **10**      **IBASIC Controller**

## Introduction

The Test Set contains an instrument controller that can run programs to control the various instruments in the Test Set and instruments/devices connected to the Test Set's external I/O ports (HP-IB, serial and parallel). Refer to "[Overview of the Test Set](#)" on page 18 for a complete description of the Test Set's hardware architecture.

The instrument controller runs a subset of the Hewlett-Packard Rocky Mountain BASIC programming language called Instrument BASIC or IBASIC. Using this programming language it is possible to develop programs which use the Test Set's instruments to automatically test a variety of radios. Software is available from Hewlett-Packard, the HP 11807 series, for testing the major radio systems currently in use today. Users can also develop their own IBASIC programs for automated radio testing.

This chapter is designed to provide the programmer with the information needed to develop IBASIC programs for use on the built-in IBASIC controller. Refer to the individual HP 11807 software manuals for information on using the IBASIC controller with Hewlett-Packard supplied software.

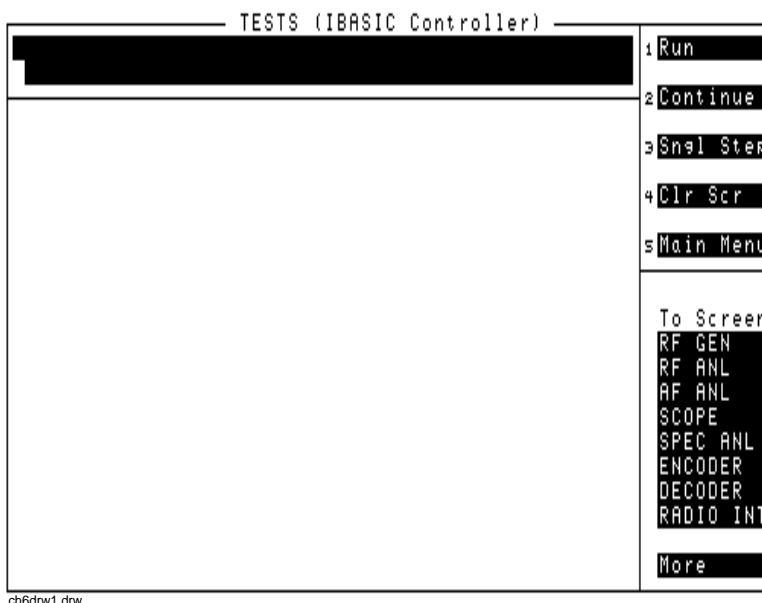


## The IBASIC Controller Screen

The Test Set has a dedicated screen for interfacing with the built-in IBASIC controller. This is the TESTS (IBASIC Controller) screen as shown in [figure 10-1](#) . This screen is accessed as follows:

- Select the front panel Menu key. The TESTS (Main Menu) screen will be displayed.
- Using the rotary knob, position the cursor on the IBASIC field in the lower center of the screen.
- Push the rotary knob and the TESTS (IBASIC Controller) screen will be displayed.

**Figure 10-1**      **The IBASIC Screen**



The TESTS (IBASIC Controller) screen can be accessed programmatically by sending the following command:

### Example

```
OUTPUT 714;"DISP TIBasic"
```

The TESTS (IBASIC Controller) screen is divided into several areas which are used by the IBASIC controller for different purposes.

The small horizontal rectangle at the top left is the IBASIC command line. As the name implies IBASIC commands can be executed from this line. Commands can be entered locally using the rotary knob or remotely using serial port 9. A maximum of 100 characters may be entered into the command line.

The vertical rectangle at the top right side is the softkey label area. The five highlighted areas within the softkey label area correspond to the five special function keys on the front panel of the Test Set. IBASIC programs can assign tables to these keys and control program execution by using ON KEY interrupts.

The large rectangle in the center of the screen is the display screen for the IBASIC controller. The IBASIC controller uses this area for, displaying alpha, numeric, and graphic information, program editing, program listing and so forth. This area operates as would the CRT on an external Hewlett-Packard 9000 Series 200/300 Workstation.

---

## Important Notes for Program Development

The Test Set is designed to operate the same way under automatic control as it does under manual control. This has several implications when designing and writing programs for the Test Set:

- To automate a particular task, determine how to do the task manually and then duplicate the steps in the program.
- In Manual Control mode, a Test Set function must be displayed and “active” to make a measurement or receive DUT data. Therefore, to make a measurement using an IBASIC program, follow these basic steps:
  1. Use the DISPlay command to select the screen for the instrument whose front panel contains the desired measurement result or data field (such as AF ANALYZER).
  2. Set the measurement field (such as SINAD) to the ON state.
  3. Trigger a reading.
  4. Read the result.

---

### NOTE

The following sections discuss developing IBASIC programs which do not use the TESTS Subsystem. Programs written for the TESTS Subsystem require the creation of supporting Library, Procedure, and Code files, and must be written using a specific program structure. The HP 11807A Radio Test Software packages are examples of this type of program.

Refer to the ["Writing Programs For the TESTS Subsystem"](#) on page 441 for information on writing programs for the TESTS Subsystem.

---

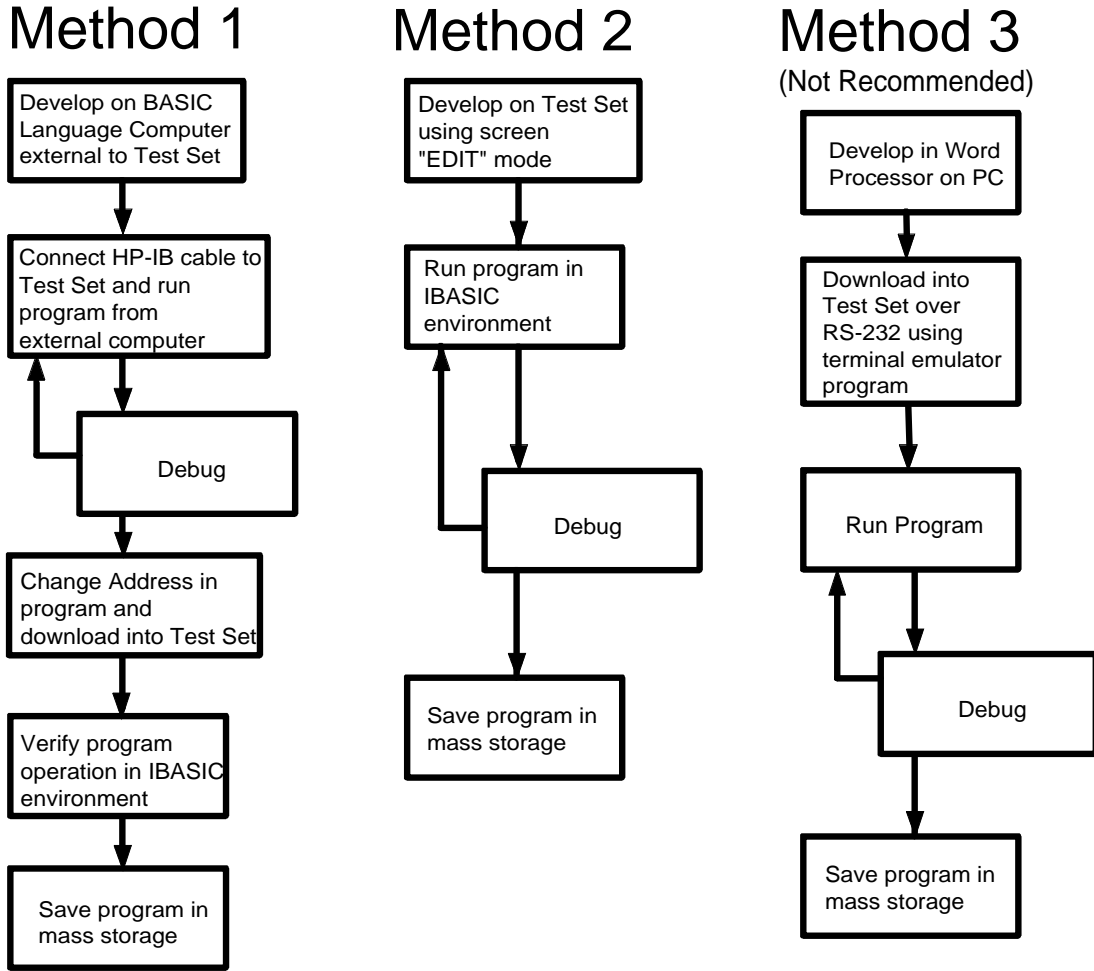
## Program Development

There are three recommended approaches for developing IBASIC programs. They are outlined in [figure 10-2](#) and discussed in more detail later in this chapter. Since the Test Set only has the rotary knob and numeric keypad for data/character entry, developing programs on the Test Set alone is not recommended. All three development methods employ an external computer or terminal. The choice of development method will typically be driven by available equipment and extent of development task. If the development task is large, it is strongly recommended that a BASIC language computer be used as outlined in development Method #1.

Method #2 is recommended for large program modification or smaller program development. Method #2 uses an external PC or terminal as the CRT and keyboard for the built-in IBASIC controller.

Method #3 is least preferred for program development or modification because no syntax checking occurs until the program is first run making it difficult to debug long programs. Details of each development method are given later in this chapter.

**Figure 10-2 Program Development Methods**



ch6drv2.drw

## Interfacing to the IBASIC Controller using Serial Ports

This section describes how to interconnect the Test Set to an external PC or terminal using the Test Set's serial I/O ports. Program development methods #2 and #3 use PC's or terminals connected to the Test Set through the Test Set's serial I/O ports. To determine which programming environment best fits your application, refer to "[Choosing Your Development Method](#)" on page 394.

### Test Set Serial Port Configuration

To prepare for IBASIC program development, the Test Set must first be configured to operate with a PC or terminal.

This includes,

- Hardware
- Cables
- Screens - I/O CONFIGURE and TESTS (IBASIC Controller)

There are two independently controllable serial interfaces in the Test Set, each using a 3-wire transmit / receive / ground implementation of the RS232 standard. The IBASIC Controller can send and receive data from either port by using its assigned select code.

#### Serial Port Information

The Test Set's rear-panel RJ-11 connector has 6 conductors. (Note that this jack appears the same as a common 4-conductor RJ-11 telephone jack, but the Test Set jack uses 6 conductors). Three of the wires are designated as Serial I/O Port address 9, and the other three wires are designated Serial I/O Port address 10 (also referred to as Serial Port B). These select codes cannot be changed.

**Serial Port 9.** Serial Port 9 is used for developing and editing IBASIC programs since it can be connected directly to the IBASIC Command Line field. It can also be used for data I/O from an IBASIC program. Settings can be changed from the I/O CONFIGURE screen, using IBASIC commands executed from the IBASIC Command Line field, or using IBASIC commands executed from an IBASIC program.

**Serial Port 10.** Serial Port 10 is primarily used for data I/O from an IBASIC program to a device-under-test- (DUT). Settings can be changed using IBASIC commands executed from the IBASIC Command Line field, or using IBASIC commands executed from an IBASIC program but not from the I/O CONFIGURE screen.

### **Reason for Two Serial Ports**

A typical application uses serial port 10 to send and receive data to and from a DUT and uses serial port 9 to print or log test results to a serial printer or PC.

In the program development environment, serial port 9 can be used to communicate with the external PC or terminal, and serial port 10 can be connected to a serial printer for generating program listings or as the destination printer for the program itself. This is schematically shown in [figure 10-4 on page 384](#). If simultaneous multiple serial I/O is not a requirement then only use serial port 9 as it can directly access the IBASIC Command Line field.

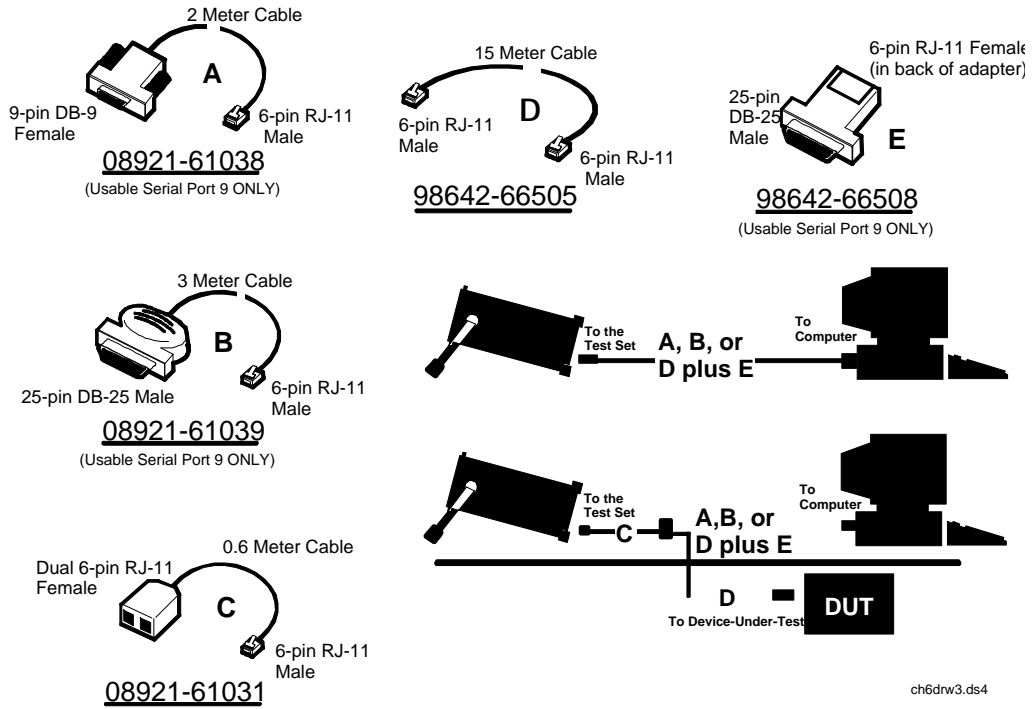
For your convenience, [figure 10-1 on page 375](#) and [Table 10-1 on page 382](#) on this page, show the cables and adapters that are available from Hewlett-Packard for connecting external devices to the Test Set's serial I/O ports. See [figure 10-4 on page 384](#) for a wiring diagram to construct your own cables. RJ-11 cables and adapters can be wired several ways. If you buy a cable or adapter other than the HP parts listed in [table 10-1](#), verify the connections for the pins indicated, before connecting the cables to the Test Set.

**Table 10-1 Available HP RS232 Serial Cables and Adapters**

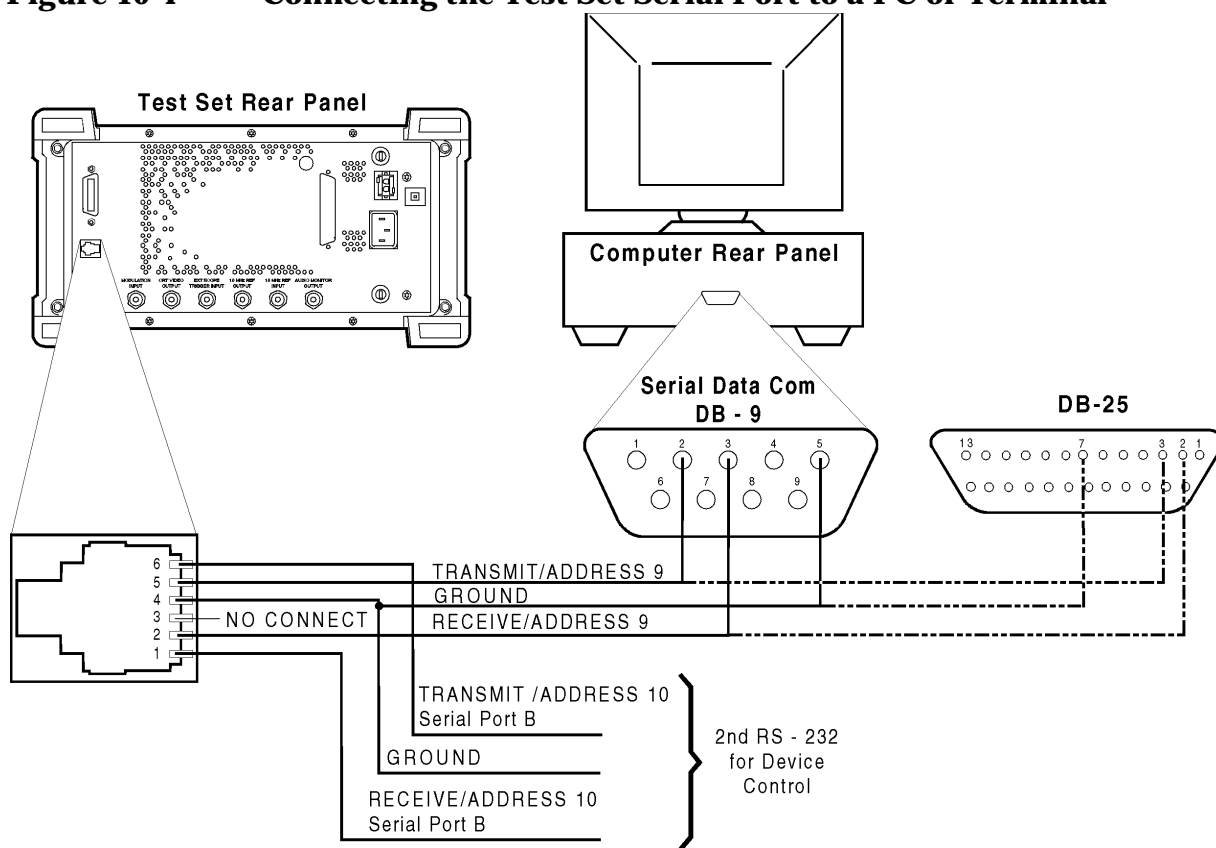
<b>Device (for RS232 Serial connections)</b>	<b>Typical Uses</b>	<b>Description</b>	<b>HP Part Number</b>
Single to Dual RJ-11 Adapter Cable	To connect to Serial Ports 9 and 10 simultaneously	Single 6-pin RJ-11 (male) to Dual 6-pin RJ-11 (female); 0.6-meter cable	08921-61031
Cable with Connectors	Test Set to PC	6-pin RJ-11 (male) to 9-pin DB-9 (female); 2-meter cable	08921-61038
Cable with Connectors	Test Set to printer or terminal	6-pin RJ-11 (male) to 25-pin DB-25 (male); 3-meter cable	08921-61039
RJ-11 to DB-25 Adapter	Use with long cable 98642-66508.	6-pin RJ-11 (female) to 25-pin DB-25 (male) Adapter	98642-66508
Cable with Connectors	Long Cable from Test Set to PC or printer (use with 98642-66508)	6-pin RJ-11 (male) to 6-pin RJ-11 (male); 15-meter cable	98642-66505



**Figure 10-3 Available HP RS-232 Serial Cables and Adapters**



**Figure 10-4** Connecting the Test Set Serial Port to a PC or Terminal



ch6drw4.drw

**Table 10-2** Port 9 or Port 10 serial cable connections

RJ-11 pins	Signal	DB-9 pins
6	Transmit/Address 10	2
5	Transmit/Address 9	2
4	Ground	5
3	not used	
2	Receive/Address 9	3
1	Receive/Address 10	3

## Serial Port 9 Configuration

**Table 10-3 on page 386** and the following paragraphs describe how to configure Serial Port 9 for communications with an external PC or terminal. Implications of the various choices are discussed.

1. Under the `To Screen` menu, select `More`, then select `IO CONFIG`.
2. The `I/O CONFIGURE` screen will be displayed.
3. Set the `Serial Baud Rate`, `Parity`, `Data Length`, `Stop Length`, `Rcv Pace` and `Xmt Pace` fields to match your PC or terminal settings. The recommended settings are shown in **table 10-3 on page 386**. These settings will be retained by the Test Set. They will not change if the `PRESET` key is pressed, if the Test Set receives a `*RST` Common Command, or the power is turned on and off.
4. Set the `Serial In` field to `Inst`. This routes Serial Port 9 to the `IBASIC Command Line` field. Characters typed on the external PC or terminal will now appear in the `IBASIC Command Line`.
5. Set the `IBASIC Echo` field to `ON`. This will cause IBASIC character output from commands (such as `LIST`, `PRINT` or `DISPLAY`) or error messages to echo characters to Serial Port 9 (the characters will in turn show up on the external PC or terminal screen). This will allow program listings and syntax error messages to be seen on the external PC or terminal.
6. Another method which can be used to output characters to the external PC or terminal is to execute the IBASIC command, `PRINTER IS 9`. This causes IBASIC to direct all print output to Select Code 9. Select Code 9 is the Test Set's Serial Port 9. Select Code 1 is the Test Set's CRT. Select Code 1 is also the default address for the `PRINTER IS` command, so all program printer output defaults to the Test Set's CRT (unless changed with the `PRINTER IS` command).
7. Set the `Inst Echo` field to `ON`. This will cause characters to be echoed back to the external PC or terminal as they are received at Serial Port 9. If the echo feature of the external PC or terminal is also enabled all the characters sent to the Test Set will be displayed twice on the external PC or terminal. Enable echo on only one device, either the Test Set or the external PC or terminal.

## Receive and Transmit Pacing

When receiving characters into the IBASIC Command Line field, the Test Set's microprocessor responds to each entry and no buffering is required. Therefore, when using your PC or terminal to send characters to the IBASIC Command Line field, it is permissible to set Rcv Pace and Xmt Pace to None.

When sending data through the Test Set's Serial Port to external devices like printers which may have small input buffers, it is important to set Rcv Pace and Xmt Pace to Xon/Xoff. This allows the printer to stop data transmission from the Test Set when the printer's buffer is full and then start it again when the printer is ready.

The Test Set has a Serial Port input buffer length of 2000 characters with firmware revision A.09.04. Buffer size becomes important when IBASIC programs expect to receive large amounts of data through the Serial Port with a single ENTER statement.

The Test Set has a Serial Port input buffer length of 2000 characters with firmware revision B.01.00 and greater. Buffer size becomes important when IBASIC programs expect to receive large amounts of data through the Serial Port with a single ENTER statement.

**Table 10-3 Test Set Serial Port 9 Configuration**

Field	Available Settings	Recommended Setting
Serial In	Inst/IBASIC	Inst
IBASIC Echo	On/Off	On
Inst Echo	On/Off	On
Serial Baud Rate	150, 300, 600, 1200, 2400, 4800, 9600, 19200	9,600
Parity	None, Odd, Even, Always 1, Always 0	None
Data Length	7 bits, 8 bits	8 bits
Stop Length	1 bit, 2 bits	1 bit
Rcv Pace (receive pacing)	None, Xon/Xoff	Xon/Xoff
Xmt Pace (transmit pacing)	None, Xon/Xoff	Xon/Xoff

## PC Configuration

To prepare for IBASIC program development, the external PC or terminal must be configured to operate with the Test Set. This configuration includes

- Hardware
- Terminal Emulator Software

### PC Serial Port Configuration

Refer to [figure 10-4 on page 384](#) for connection details. Connect the Test Set's Serial Port 9 to a serial I/O (input/output) port on the PC. On many PCs, a serial port is available as either a 25-pin DB-25 (female) connector or a 9-pin DB-9 (male) connector. This port can be configured as COM1, COM2, COM3, or COM4 (communications port 1, 2, 3, or 4) depending on the installed PC hardware and user-defined setup. Refer to the instructions shipped with the PC for hardware and software configuration information.

### Terminal Emulator Configuration Information

A “terminal emulator” is an application program running on the PC that communicates with one of the serial communication ports installed in the PC. It provides a bi-directional means of sending and receiving ASCII characters to the Test Set's serial port.

In general, a “terminal emulator” enables the PC to act like a dedicated computer terminal. This type of terminal was used before PCs to allow remote users to communicate through RS232 with central mainframe computers. An ANSI-compatible terminal like the Digital Equipment Corporation VT-100 can be used to directly communicate with the Test Set. PC terminal emulation application programs have been designed to have setup fields much like these older technology terminals.

### **Setting Up Microsoft Windows Terminal on your PC (Windows Version 3.1) .**

1. Start the Terminal program in Windows.
2. From the Terminal Menu select Settings then Emulation.
3. Select DEC VT-100 (ANSI)
4. From the Terminal Menu select Settings then Terminal Preferences
5. Edit the Terminal Preference settings to match the following
  - Terminal Modes
    - Line Wrap: Off
    - Local Echo: Off
    - Sound: Off
  - Columns: 132
  - CR->CR/LF
    - Inbound: Off
    - Outbound: Off
  - Cursor
    - Block
    - Blink: On
  - Terminal Font: Fixedsys
  - Translations: None
  - Show Scroll Bars: On
  - Buffer Lines: 100
  - Use Function, Arrow, and Ctrl Keys for Windows: Off
6. From the Terminal menu select Settings then Text Transfers.
7. Edit the Text Transfer settings to match the following.
  - Flow Control: Standard Flow Control
  - Word wrap Outgoing Text at Column: Off
8. From the Terminal menu select Settings then Communications
9. Edit the Communications settings to match those of the Test Set. An example setting is shown below.
  - Baud Rate: 9600
  - Data Bits: 8

- Stop Bits: 1
- Parity: None
- Flow Control: Xon/Xoff
- Connector: Com1 (be sure to match your current setup)
- Parity Check: Off
- Carrier Detect: Off

### **Setting Up ProComm Revision 2.4.3 on your PC .**

ProComm is a general purpose telecommunications software package for PC's with MS-DOS. One of its functions is to provide an RS-232 terminal function on a typical PC.

**Running ProComm in MSDOS.** (You can use ProComm's built-in help function to learn more about setting it up).

1. To access the help and command functions, press the Alt and F10 keys simultaneously (abbreviated as Alt+F10).
2. Press the space bar to move among the choices for a particular field.
3. Press ENTER to accept the displayed choice.

### **Setting up the ProComm Software.**

1. Press Alt+ P to access the LINE SETTINGS window.
2. Enter the number 11. This will automatically set the following:
  - Baud rate: 9600
  - Parity: None
  - Data Bits: 8
  - Stop Bits: 1
  - Selected communications port: COM1 (This may be different on your PC)
3. To select a different communications port, enter the following numbers:
  - 20: COM1
  - 21: COM2
  - 22: COM3
  - 23: COM4
4. Enter the number 24 to save changes, to make the new configuration your default, and to exit LINE SETTINGS.
5. Press Alt+S for the SETUP MENU.

6. Enter the number 1 for MODEM SETUP.
7. Enter the number 1 for the Modem init string.
8. Press Enter to set a null string.
9. Press Esc to exit MODEM SETUP back to the SETUP MENU.
10. Enter the number 2 for TERMINAL SETUP.
11. Terminal emulation: VT-100
  - Duplex: FULL
  - Flow Control: XON/XOFF
  - CR translation (in): CR
  - CR translation (out): CR
  - BS translation: NON-DEST
  - BS key definition: BS
  - Line wrap: ON
  - Scroll: ON
  - Break length (ms): 350
  - Enquiry (CNTL-E): OFF
12. Press Esc to exit Terminal Setup back to the Setup Menu.
13. Enter the number 4 for General Setup.
  - Translate Table: OFF
  - Alarm sound: OFF
  - Alarm time (secs): 1
  - Aborted downloads: KEEP
14. Press Esc to exit General Setup back to the Setup Menu.
15. On the Setup Menu, press S to save your entries.
16. Press Esc to exit the Setup Menu.
17. Press Alt+X to exit ProComm back to MS-DOS.



### **Setting Up HP AdvanceLink (HP 68333F Version B.02.00) on your PC.**

HP AdvanceLink is a software program which allows PCs to be used as an alphanumeric or graphics terminal. It can also automate terminal and file-transfer functions. The version described will work with PCs with the MS-DOS or PC-DOS operating systems. (AdvanceLink for Windows is also available, and configuration is very similar).

#### **Running AdvanceLink in MSDOS.**

1. Press the Tab key to move from one field to the next, which also accepts the displayed choice.
2. Press the NEXT CHOICE and PREVIOUS CHOICE keys to move among the choices for a particular field.

#### **Setting up the AdvanceLink Software.**

1. Press the TERMINAL function key.
2. Press CONFIG KEYS.
3. Press GLOBAL CONFIG.
  - Keyboard: USASCII
  - Personality: ANSI
  - Language: ENGLISH
  - Terminal Mode: Alphanumeric
  - Remote To: enter your PC's selected serial port number, often, Serial 1
  - Printer I/F: None
  - Memory Size: 32K
  - Plotter I/F: None
  - Video Type: select your display type
  - Forms Path: no entry
  - Screen Size: select your size – 23 or 24
4. Press DONE to return to the Config screen.

5. Press REMOTE CONFIG (to set up the Serial port you selected above in Remote To).
  - Baud Rate: 9600
  - Parity/DataBits: None/8
  - Enq Ack: NO
  - Asterisk: OFF
  - Chk Parity: NO
  - SR(CH): LO
  - Recv Pace: Xon/Xoff
  - CS(CB)Xmit: NO
  - XmitPace: Xon/Xoff
6. Press DONE to return to the Config screen.
7. Press TERMINAL CONFIG.
  - Terminal Id: 2392A
  - LocalEcho: OFF
  - CapsLock: OFF
  - Start Col: 01
  - Bell: ON
  - XmitFnctn(A): NO
  - SPOW(B): NO
  - InhEolWrp(C): NO
  - Line/Page(D): LINE
  - InhHndShk(G): NO
  - Inh DC2(H): NO
  - Esc Xfer(N): YES
  - ASCII 8 Bits: YES
  - Fld Separator: down arrow or US
  - BlkTerminator: up arrow or RS
  - ReturnDef: musical note or CR
  - Copy: Fields
  - Type Ahead: NO
  - Row Size: 160

- Host Prompt Character: left arrow or D1
  - Horiz. Scrolling Increment: 08
8. Press DONE to return to the Config screen.
  9. Press DONE to return to the Terminal screen.
  10. Press MAIN to return to the Main screen.
  11. Press EXIT ADVLINK to exit.

## Terminal Configuration

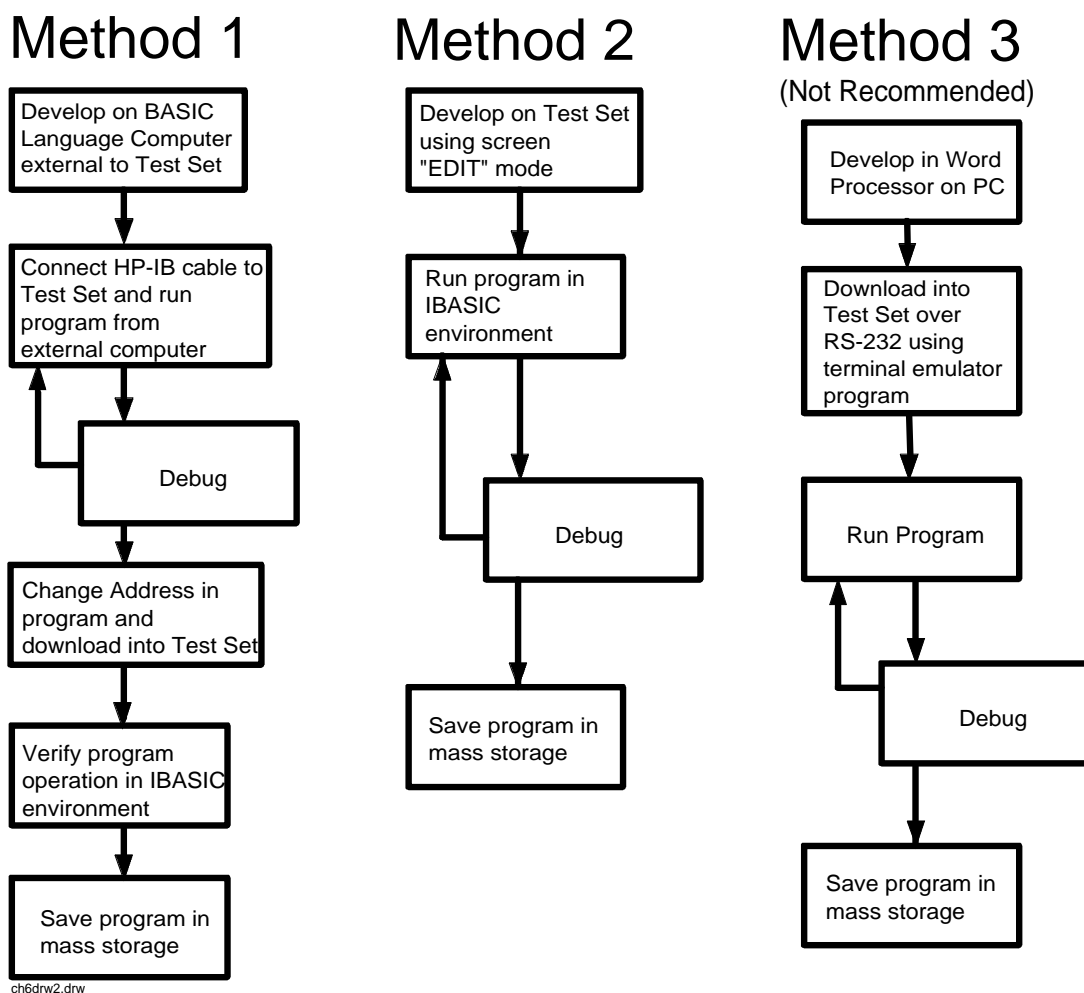
Use the cable information in [table 10-1 on page 382](#) and [figure 10-1 on page 375](#) for connecting to an external terminal. Terminals typically have a DB-25 (male) connector. Set the terminal for DEC VT-100 ANSI emulation. Many ASCII terminals will also function properly.

To set up the terminal, use the field settings found in the HP AdvanceLink terminal emulator section found earlier in this chapter. As a minimum, make sure the terminal's basic setup information matches the fields on the Test Set's I/O CONFIGURE screen (refer to [table 10-3 on page 386](#) for recommended settings).

## Choosing Your Development Method

There are three fundamental methods for developing IBASIC programs for the Test Set. See [figure 10-5](#) below.

**Figure 10-5** Three Possible Development Methods



### Method 1

Using a BASIC language computer (either an HP technical computer or a PC running BASIC with HP-IB) is the best method for developing any size program. This is because the program can be debugged directly on the external computer before downloading the program into the Test Set. Using this approach the programmer can observe the Test Set's display to see changes in state and easily verify the correct measurements.

## Method 2

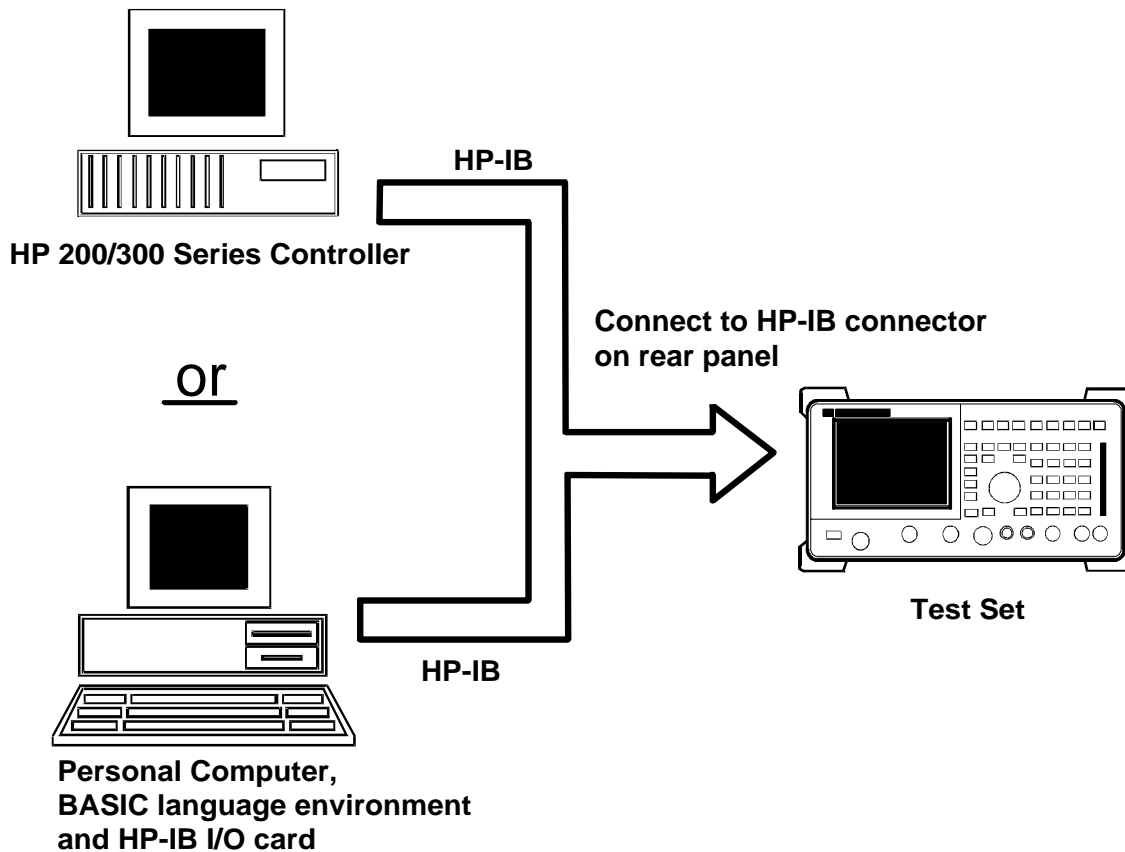
If a BASIC language computer is not available, program development can be done directly on the Test Set using the IBASIC EDIT mode. A PC connected to the Test Set through RS-232, as described earlier in this chapter, is used as the CRT and keyboard for the internal controller. In this method, the program always resides in the Test Set and can be run at any time. Mass storage is usually an SRAM card. When running IBASIC programs on the Test Set's internal controller, the Test Set displays only the IBASIC screen, not the individual instrument screens as the program executes. This makes troubleshooting larger programs more difficult.

## Method 3

The third method of program development is to use a word processor on a PC with RS-232, and then download the program into the Test Set for execution. This is the least favorable choice for development because downloading code into the Test Set over RS-232 requires a loader utility program running in the Test Set and a RAM memory card present as an intermediate storage location before running the program. (For shorter programs, the intermediate storage location is not necessary.) No IBASIC command syntax is checked until the program is run after downloading. Also, when running IBASIC programs on the Test Set's internal controller, the Test Set displays only the IBASIC screen, not the individual instrument screens as the program executes. This makes troubleshooting larger programs more difficult.

## Method #1. Program Development on an External BASIC Language Computer

Figure 10-6 Connecting IBASIC Language Computers to the Test Set



ch6drw5.drw

## Configuring the Test Set's HP-IB Interface

To use HP-IB (the IEEE 488 interface bus) as a means of communicating with the Test Set, connect a standard HP-IB cable (such as the HP 10833B) between the Test Set's rear-panel HP-IB connector and the HP-IB connector on the external BASIC language computer.

- On the Test Set
1. Select the I/O CONFIGURE screen.
  2. Set the Mode field to Talk&Lstn.

---

**NOTE**

If the Mode field is set to Control, there could possibly be a System Controller conflict between the external BASIC language computer and the Test Set, resulting in either an Interface Status Error or "lock up" of the HP-IB. Refer to "[Passing Control](#)" on page 337.

3. Set the HP-IB Adrs field to the desired address for the Test Set. The default value is 14.

## Compatible BASIC Language Computers

As shown in [figure 10-6 on page 396](#), there are two types of computers that can be used in this development method.

- The HP 9000 Series 200/300 Workstation running HP Rocky Mountain BASIC 6.2 or later. IBASIC is a subset of HP Rocky Mountain BASIC (RMB). All IBASIC commands are compatible with RMB and thus will execute from a HP 9000 Series 200/300 Workstation.
- A PC, running Windows 3.1 or Windows NT, with HP BASIC for Windows and an HP-IB interface card can be used.

## HP BASIC for Windows PC Configuration for Windows NT Operating System

To prepare for HP BASIC program development utilizing Windows NT, the external PC must be configured to operate with the Test Set.

You will need:

- an HP 82341B/C interface card (the HP 82335 card does not support Windows NT)
- a licensed version with security key of HP BASIC for Windows. You can get the latest files from <ftp://hpslsup.lvld.hp.com/pub/pc/index.html>. Select the HP BASIC for windows archives and follow the instructions for download.

How to install:

1. Install the HP 82341B/C into an open expansion slot. Refer to the interface card's installation guide details. Utilize the default card settings.
2. Install the SICL libraries using the SETUP32.EXE setup file.
3. Run the SICL I\_O Config program to configure the card.
4. Select HP 82340/82341 HP-IB from the available interface list of choices.
5. Select the Configure command button. Use the default settings shown by the program. This will verify that the card is functioning.
6. Install HP BASIC for Windows 6.3 or later.
7. Run HP BASIC for Windows.
8. Edit the AUTOST file, change line 330 as follows:

Example

```
LOAD BIN "HPIBS;DEV hpib7" !SICL Library for HP 82341 card
```

9. Re-store the AUTOST file
10. Quit HP BASIC for Windows
11. Run HP BASIC for Windows. The program should load normally and allow you to send orders to the Test Set.

## Program Development Procedure

As discussed in "[Overview of the Test Set](#)" in [chapter](#) , the Test Set has two HP-IB buses, an internal HP-IB at select code 8 and an external HP-IB at select code 7. The Test Set's built-in IBASIC controller uses the internal HP-IB to communicate with the Test Set's various instruments and devices. The process of developing a program on an external BASIC language computer utilizes this hardware feature to an advantage.



First, develop the program directly on the external BASIC language computer treating the Test Set as a device on the external BASIC language computer's HP-IB. For example, to setup the Test Set's RF Generator use the OUTPUT command with the Test Set's HP-IB address. If the select code of the HP-IB card in the external BASIC language computer is 7 and the address of the Test Set is 14 the address following the OUTPUT command would be 714. When the command executes on the external BASIC language computer the information on how the Test Set's RF Generator is to be configured is sent to the Test Set through its external HP-IB bus. After the program is fully developed, making it run on the Test Set is simply a matter of changing the address of all the HP-IB commands to 8XX (Test Set internal HP-IB bus) and downloading the program into the Test Set's IBASIC controller and executing it.

There are two ways of allowing easy conversion of all HP-IB commands to a different address. The first way is to establish a variable to which the 3-digit address number is assigned.

For example

#### Example

```
10 Addr = 714 ! Sets the value of variable Addr to be 714.
20 OUTPUT Addr;"*RST"!Commands the Test Set to reset at address 714.
```

To change the address, simply change the value of variable Addr to 814.

For example

#### Example

```
10 Addr = 814 ! Sets the value of variable Addr to be 814.
20 OUTPUT Addr;"*RST"! Commands the Test Set to reset at address 814.
```

A second method is to assign an I/O path to the desired I/O port.

For example

To control device #14 on the port with select code 7.

#### Example

```
20 ! Establishes IO path to select code 7 address 14
10 ASSIGN @Device TO 714
30 ! Commands Test Set to reset at address 714.
20 OUTPUT @Device;"*RST"
```

To change the address, simply change line 10 to

**Example**

```
10 ASSIGN @Device TO 800.
```

**NOTE**

The dedicated HP-IB interface at select code 8 conforms to the IEEE 488.2 Standard in all respects but one. The difference being that each instrument on the bus does not have a unique address. The Instrument Control Hardware determines which instrument is being addressed with the command syntax. As such an explicit device address does not have to be specified. The address 800 and 814 are equally correct.

## Downloading Programs to the Test Set through HP-IB

An IBASIC PROGRAM subsystem has been developed to allow the external BASIC language controller to download programs to the Test Set through HP-IB (refer to the "[PROGRAM Subsystem Commands](#)" on page 422 for more information on the PROGRAM Subsystem). Four commands from the external BASIC language controller to the Test Set are necessary to transfer the program. The commands are executed serially allowing enough time for each command to finish executing. (The Test Set's HP-IB Mode field must be set to Talk&Lstn, and the TESTS (IBASIC CONTROLLER) screen must be displayed).

1. OUTPUT 714;"PROG:DEL:ALL"
  - Deletes any programs that reside in Test Set RAM.
2. OUTPUT 714;"PROG:DEF #0"
  - Defines the address in Test Set RAM where the downloaded program will be stored.
3. LIST #714
  - Causes all program lines to transfer over HP-IB to the Test Set which is at address 714.
4. OUTPUT 714;" "END
  - Defines end of download process by generating an EOI command.

After the above commands complete the program code will be in the Test Set ready to run. If any bugs are detected when the program is run, the program can be uploaded back into the external BASIC language controller to correct the error. Alternately the full screen IBASIC EDIT function through RS-232 can be used to correct the error (refer to "[Method #2. Developing Programs on the Test Set Using the IBASIC EDIT Mode](#)" on page 403 for details).

After the program is working properly in the Test Set IBASIC environment, it should be stored for backup purposes.

## Uploading Programs from the Test Set to an External BASIC Controller through HP-IB

To upload a program from the Test Set to an external BASIC language controller through HP-IB the following program, which uses a command from the PROGram subsystem to initiate the upload, must be running on the external BASIC language controller. The uploaded program is stored to a file specified by the user.

In the following program the external BASIC language controller is a PC running TransEra HT BASIC. The file is stored to the C:\HTB386 directory. If the external BASIC language controller is an HP 9000 Series 200/300 Workstation, modify the mass storage volume specifier appropriately. After running the program, the uploaded program code will be in the designated file. Use the GET command to retrieve the file for editing.

### Example

```

10      ! PROGRAM TO UPLOAD IBASIC CODE FROM TEST SET TO BASIC CONTROLLER
        THROUGH HP-IB.
20
!#####
30      !
40      ! The file for uploaded code will be "C:\htb386\code".
50      ! If you want to use a different file or directory, modify the two
        lines
60      ! with the labels "File_name_1" and "File_name_2".
70      !
80
!#####
90      Addr=714                                !Test Set HP-IB address
100     ALLOCATE Line$(200)
110     PRINTER IS 1
120     CLEAR SCREEN
130     DISP "It may be several minutes before code begins transferring if
        the program is long"
140     OUTPUT Addr;"*RST"                        !Reset the Test Set
150     OUTPUT Addr;"DISP TIB"                  !Displays the IBASIC screen
160     OUTPUT Addr;"PROG:EXEC 'CLS'"           !Clears the Test Set display
170     OUTPUT 714;"PROG:DEF?"                 !Initiates the upload of whole program
180     ENTER Addr USING "X,D,#";Count_len!Number of lines in program
190     ENTER Addr USING VAL$(Count_len)&"D,#";Char_count !Number of
        characters
200     !
210     File_name_1: CREATE ASCII "C:\htb386\code", (1.05*Char_count/256)+5
220     ! Number of records reserved for upload.
230     File_name_2: ASSIGN @File TO "C:\htb386\code"
240     !
250     DISP "Transferring code from Test Set"
260     LOOP                                     !Program transfer loop.
270     ENTER Addr;Line$ !CR/LF terminates each line.
280     PRINT Line$                               !Displays new lines on Test Set display.
290     OUTPUT @File;Line$!Transfer new line to file.
300     Char_count=Char_count-LEN(Line$)-2      !Reduces Char_count by the
        number of
310                                           ! characters in current line.
320     EXIT IF Char_count<=0
330     END LOOP
340     !
350     ASSIGN @File TO *!Cleans out file buffer.
360     ENTER Addr;Line$!Close off reading
370     CLEAR SCREEN

```

```
380  DISP "Transfer complete."  
390  LOCAL Addr  
400  END
```

---

## Method #2. Developing Programs on the Test Set Using the IBASIC EDIT Mode

If a BASIC language computer is not available, program development can be done directly on the Test Set using the IBASIC EDIT mode. A terminal or PC connected to the Test Set through RS-232 is used as the CRT and keyboard for the Test Set's built-in IBASIC controller. In this method, the program always resides in the Test Set and can be run at any time. Mass storage is usually an SRAM memory card. When running IBASIC programs on the Test Set's internal controller, the Test Set displays only the IBASIC screen.

The Test Set's IBASIC controller has an editor that is interactive with a terminal or PC over the RS-232 serial port. (The editor does not work unless a terminal or PC with terminal emulator is connected to Serial Port 9.) The editor, hereafter referred to as the "IBASIC EDIT Mode", allows the programmer to develop code directly in the Test Set with no uploading or downloading. The IBASIC EDIT Mode can be used to develop programs from scratch or to modify existing programs. Refer to "[Interfacing to the IBASIC Controller using Serial Ports](#)" on page 380 for information on connecting a terminal or PC to the Test Set.

### Selecting the IBASIC Command Line Field

To use the IBASIC EDIT Mode for program development, the IBASIC Command Line field must be displayed on the Test Set and Serial Port 9 must be connected to the IBASIC Command Line field. An IBASIC command, sent as a series of ASCII characters through Serial Port 9, will appear on the IBASIC Command Line field. When a carriage return/line feed is encountered, the Test Set will attempt to execute the command. To display the IBASIC Command Line field on the Test Set execute the following steps:

1. Press the TESTS key.
2. The TESTS (Main Menu) screen will be displayed.
3. Using the rotary knob, position the cursor on the IBASIC Cntrl field and select it.
4. The TESTS (IBASIC CONTROLLER) screen will be displayed.
5. The small horizontal rectangle at the top-left is the IBASIC Command Line.

### **To Access the IBASIC Command Line Field**

1. Position the cursor on the screen's upper left. This is the IBASIC Command Line field.
2. The IBASIC Command Line field does not have a title like other fields in the Test Set; it is the highlighted, horizontal 2-line "bar" just below the screen title, TESTS (IBASIC Controller).

### **To Use the IBASIC Command Line Field with the Test Set's Rotary Knob**

1. Position the cursor at the IBASIC Command Line field and push the knob.
2. A Choices: field will be displayed in the lower, right corner of the display.
3. By rotating the knob, a list of ASCII characters and cursor positioning commands can be displayed on the right side of the screen.
4. When the cursor is next to the desired character or command, push the knob to select that character.
5. No external hardware is required for this entry method, but it is tedious and is recommended only for short commands. Use this method when doing simple tasks such as initializing memory cards or CATaloging a memory card.
6. Program development using the rotary knob alone is not recommended.

## **Entering and Exiting the IBASIC EDIT Mode**

To enter the IBASIC EDIT Mode first position the cursor on the IBASIC Command Line field, type the word EDIT on the terminal or PC connected to the Test Set and then press the ENTER key on the terminal or PC. At this point the Test Set will fill the PC screen with 22 lines of IBASIC code from the program currently in the Test Set's RAM memory. No program lines will be displayed on the Test Set screen. If no program is currently in the Test Set's memory, the number 10 will be displayed on the terminal or PC screen. This represents program line number 10 and is displayed to allow you to begin writing an IBASIC program beginning at line number 10. The "\*" annunciator will be displayed in the upper, right corner of the Test Set indicating that the IBASIC controller is running to support the full screen edit mode.

After editing is complete, exit the IBASIC EDIT Mode by pressing the terminal or PC's ESCAPE key twice or pressing the SHIFT CANCEL keys on the Test Set.

A variety of editing commands are supported by the IBASIC EDIT Mode. These commands are activated in the Test Set as escape code sequences. Most terminals and PC terminal emulator programs allow function keys to be configured with user defined escape code sequences and user defined labels for the keys. An escape command (when received by a peripheral device like a printer or the Test Set) causes the peripheral to recognize subsequent ASCII characters differently. In the case of the Test Set, escape sequences are used for executing IBASIC EDIT Mode editing commands.

For example, ESCAPE [L causes the Test Set to insert a new line number where the cursor is positioned. [Table 10-4 on page 406](#) lists the editing escape codes for the Test Set. There is no escape code for DELETE CHARACTER. Use the Backspace key for deleting. Use the arrow keys to position the cursor.

## Setting Up Function Keys In Microsoft Windows Terminal

When in the TERMINAL mode, click on Settings, then Function Keys. `^[` is ESCAPE in Windows Terminal. See [table 10-4 on page 406](#) for the escape codes.

---

### NOTE

Windows Terminal seems to work best when a mouse is used to access the function keys, not the keyboard. Also, scrolling a program works best when the Terminal window display is maximized).

---

## Setting Up Function Keys in HP AdvanceLink

- From the `Main` (highest level) screen, set up the 8 softkeys as follows:
  1. Display User Definition screens by pressing `Ctrl F9`.
  2. Enter all the LABEL titles for K1 through K8.
  3. Activate the “Display Function” feature by pressing softkey `F7`.
  4. Now you can enter the escape codes for each edit command aligned with the soft key definitions you just entered. With the Display Functions key pressed, when you press the escape key, a left arrow will be displayed.
- Once you have set up all 8 keys, you activate them by pressing `Shift F12`. To deactivate your user defined softkeys, press `F12`.
- (
  -

is ESCAPE in HP AdvanceLink. See [table 10-4 on page 406](#) for the escape codes.

## Setting Up Function Keys in ProComm

ProComm does not have function keys. However, escape sequences can be assigned to number keys 0 through 9 by using the Keyboard Macro function. This function is accessed by keying Alt+M. There is no method of displaying key labels so they will have to be recorded elsewhere. See the ProComm manual for further information.

**Table 10-4**      **Edit Mode Escape Code Commands**

<b>Function Key Names</b>	<b>Windows Terminal Escape Codes</b>	<b>HP AdvanceLink Escape Codes</b>
INSERT LINE	^[[L	(-[L
DELETE LINE	^[[M	(-[M
GO TO LINE	^[g	(-g
CLEAR LINE	^[[K	(-[K
PAGE UP	^[OQ	(-OQ
PAGE DOWN	^[OR	(-OR
RECALL LINE	^[r	(-r
BEGIN LINE	^[OP	(-OP
END LINE	^[OS	(-OS



## **Method #3. Developing Programs Using Word Processor on a PC (Least Preferred)**

The third method of IBASIC program development is to write the program using a word processor on a PC, save it as an ASCII file, and then download it into the Test Set through the serial port. The benefit of this method is that it can be done on the PC without connecting to a Test Set until download and no BASIC language compiler/interpreter is needed. The primary drawback is that no syntax checking occurs until the downloaded program is run on the Test Set. A second drawback is that, especially for longer programs (>100 lines), it is very time-consuming to transfer the code into the Test Set.

### **Configuring a Word Processor**

The word processor on which the IBASIC code is developed must be able to save the file in ASCII format and have an ASCII file transfer utility. This is necessary because word processors use a variety of escape codes to mark all the special display formats such as bold face, font size, indented text, and the like. When a word processor file is stored in ASCII format, all escape codes are stripped off. The ASCII file transfer utility is used to transfer the file to the Test Set.

---

**NOTE**

The GET command can be used on external BASIC language controllers to load ASCII files containing IBASIC programs developed on word processors. Once loaded, the steps for downloading described in "[Method #1. Program Development on an External BASIC Language Computer](#)" on page 396 can be used to transfer the program to the Test Set.

---

## Writing Lines of IBASIC Code on a Word Processor

When writing IBASIC programs, follow these steps to ensure that the Test Set will accept the code when it is downloaded.

1. Always begin new lines at the far left margin. Never use a leading space or tab.
2. Number each consecutive line just like an IBASIC language program.
3. Typically begin with 10 and increment by ten for each consecutive line.
4. Do not leave any space or double space between lines.
5. Make sure to use hard carriage return / line feeds at the end of each line.
6. When saving the completed program, save it as an ASCII file. Some word processors have ASCII options which require that the user specify CR/LF at the end of each line. It is important that each line end with a carriage return / line feed.
7. Experiment with a short program first to make sure everything is working correctly.

## Transferring Programs from the Word Processor to the Test Set

For short (less than 100 lines) programs, use an ASCII file transfer utility on the PC to send the program, one line at a time, down to the Test Set over RS-232 directly into the IBASIC Command Line field. The Test Set must be configured to receive serial ASCII characters by positioning the Test Set cursor at the IBASIC Command Line field as explained under "[Method #2. Developing Programs on the Test Set Using the IBASIC EDIT Mode](#)" on page 403. With this setup, when ASCII characters are received they are sent to the IBASIC Command Line field. When a carriage return / line feed is received, the Test Set will parse the line into the IBASIC program memory. Each line takes about two seconds to scroll in and be parsed. This becomes very time consuming for long programs. An alternative for longer programs is discussed later in this section.

To start the transfer process make sure there is no program in the Test Set's IBASIC RAM memory by executing a SCRATCH command from the IBASIC Command Line.

The following example shows how to transfer a short program (<100 lines) using Microsoft Windows Terminal.

1. Make sure the Test Set cursor is in the upper left of the IBASIC Command Line field.

2. Select the Terminal application in the Accessories Group. Set it up as described in earlier in this chapter.
3. Select the following:
  - Settings
    - Text Transfers
    - Flow Control: Line at a Time
    - Delay Between Lines: 25/10 Sec
    - Word Wrap
    - Outgoing Text at Column: Off.
4. Select the following:
  - Transfers
    - Send Text File
    - Following CR:
      - Strip LF selected
      - Append LF not selected.
5. Select the text file to be transferred and begin the transfer by selecting (OK).

As the transfer starts the IBASIC Command Line field will intensify and characters will scroll in left to right. As each line is finished the "\*" annunciator will be displayed, for about 0.5 seconds, in the upper, right corner of the Test Set indicating that the IBASIC controller is running as the line is parsed. If another line is sent before this parsing is complete, the Test Set will beep indicating an error, and the next line of the transfer will be rejected.

If the transfer is rejected, the transfer must be halted and the delay between lines increased to a slightly higher number. Start the transfer again from the beginning. When all lines have transferred, list the program to verify it was completely received. At this time, the program is ready to run. The RUN command can be keyed in from the PC or the K1 Run key in the TESTS (IBASIC Controller) screen can be pressed.

---

**NOTE**

Do not press the Run Test key in the TESTS (Main Menu) screen as this will scratch the program you just loaded and look to the memory card for a procedure file.

---

For longer programs (greater than 100 lines), transferring the ASCII text file directly into the IBASIC program memory through the RS-232 serial port is too time consuming. To speed the process up, it is necessary to transfer the program using a two step process.

1. Transfer the ASCII text file directly to a Test Set mass storage location (typically an SRAM card).
2. Perform a GET command to bring the program from mass storage into the IBASIC program memory.

To perform the ASCII text file transfer for long programs, an IBASIC program, running in the Test Set, is required to manage the transfer. A suitable program titled "ASCII\_DN" (for ASCII downloader) is shown on the following page.

The ASCII\_DN program runs on the Test Set and directs ASCII characters coming in Serial Port 9 directly to a file named TEMP\_CODE on an SRAM card. The program creates the TEMP\_CODE file on the SRAM card with a size of 650 records (166 Kbytes or enough for about 6600 lines of ASCII text). When the program is run, it displays Ready to receive ASCII file data. When this prompt is displayed, initiate the transfer of the ASCII text file representing the program from the PC to the Test Set. Shown below are two methods of sending an ASCII file from the PC to the Test Set. Both methods require that the ASCII\_DN program be running in the Test Set when the transfer begins. The ASCII\_DN program can be transferred into the Test Set either by typing it in using the IBASIC EDIT Mode described earlier, or downloading it from an ASCII text file one line at a time as explained earlier.

### Example

```

10 ! ASCII_DN
20 ! Program to download ASCII program file from PC to the Test Set
   through RS-232
30 !
#####
40 !
50 ! This program must be loaded into the Test Set and run on the Test
   Set.
60 ! It directs ASCII characters that come in the Serial Port 9 to a
   file
70 ! named "TEMP_CODE" on an SRAM card. After the transfer is complete,
80 ! you must SCRATCH this program and GET the transferred program from
90 ! the "TEMP_CODE" file.
100 !
110 !
#####
120 COM /File_name/ File_name${10}
130 DIM In${200}
140 File_name$="TEMP_CODE" !File name on RAM card
150 CLEAR SCREEN
160 CLEAR 9!Clears Test Set serial bus
170 OUTPUT 800;"*RST"
180 ! Set up Test Set Serial Port 9 to receive ASCII text file
190 OUTPUT 800;"CONF:SPORT:BAUD '9600';PAR 'None';DATA '8 Bits'"
200 OUTPUT 800;"CONF:SPORT:STOP '1 Bit';RPAC 'Xon/Xoff';XPAC
   'Xon/Xoff'"
210 OUTPUT 800;"CONF:SPORT:SIN 'IBASIC';IBECHO 'OFF'"
220 CALL Code(File_name$,In$)

```

## Method #3. Developing Programs Using Word Processor on a PC (Least Preferred)

```

230 END
240 Purge_it:SUB Purge_it!Purges File_name on card
250   COM /File_name/ File_name$
260   OFF ERROR
270   PURGE File_name$&":INTERNAL"
280   SUBEND
290 Code:SUB Code(File_name$,In$)
300   ON ERROR CALL Purge_it      !Branches if CREATE statement
returns error
310   CREATE ASCII File_name$&":INTERNAL",650      !Creates file on
card
320   OFF ERROR
330   ASSIGN @File TO File_name$&":INTERNAL"
340   PRINT TABXY(1,5);"Ready to receive ASCII file data."
350   PRINT
360 Begin:ON TIMEOUT 9,1 GOTO Begin      !Loops until data begins coming
370   ENTER 9;In$
380   OUTPUT @File;In$
390   PRINT In$
400 Transfer:LOOP      !Loops to bring in ASCII file one line at a time
410   ON TIMEOUT 9,5 GOTO Done      !Exit loop if data stops for >5
sec.
420   ENTER 9;In$
430   PRINT In$
440   OUTPUT @File;In$
450   END LOOP
460 Done:ASSIGN @File TO *
470   CLEAR SCREEN
480 ! Returns Test Set Serial Port 9 input to "instrument" allowing
serial
490 ! communication to the IBASIC Command line field.
500   OUTPUT 800;"CONF:SPORT:SIN 'Inst';IECHO 'ON';IBECHO 'ON'"
510   PRINT TABXY(1,5);"Down load of ASCII file is complete."
520   SUBEND

```

## Sending ASCII Text Files Over RS-232 With Windows Terminal

Set up the Windows Terminal emulator software on the PC as covered in "[Setting Up Microsoft Windows Terminal on your PC \(Windows Version 3.1\)](#)" on [page 388](#). Load and run the ASCII\_DN download program in the Test Set's IBASIC controller. When the prompt `Ready to receive ASCII file data` is displayed on the Test Set, make the following settings in Windows Terminal on the PC:

1. Select `Settings`.
2. Select `Text Transfers`.
3. Select `Flow Control: Standard Flow Control`.
4. Select `Word Wrap Outgoing Text at Column: unselected`.
  - This will use `Xon/Xoff` flow control by default.
5. Select `OK`.
6. Select `Transfers`.
7. Select `Send Text File`.
8. Set `Strip LF off` and `Append LF off`. (It is important that the line feeds that are in the ASCII file not be stripped or the file transfer will not work).
9. Select or enter the file name to transfer.
10. Begin the transfer by selecting `OK`.

At this point, each line of the program will rapidly scroll across the screen of the Test Set. When the transfer is finished, the prompt `Download of ASCII file complete.` will be displayed on the Test Set.

Before running the downloaded program, execute a `SCRATCH` command on the IBASIC Command Line to remove the ASCII\_DN download program from Test Set memory.

Next, execute a `GET TEMP_CODE` command on the IBASIC Command Line. This will load the ASCII text into the IBASIC program memory.

Finally, execute a `RUN` command on the IBASIC Command Line. This will run the program. If any syntax errors are present in the program IBASIC will generate the appropriate error messages.

## Sending ASCII Text Files over RS-232 with ProComm Communications Software

Set up the ProComm terminal emulator software on the PC as covered in "[Setting Up ProComm Revision 2.4.3 on your PC](#)" on page 389. On the Test Set, enter and run the ASCII\_DN download program in the IBASIC controller. When the prompt `Ready to receive ASCII file data` is displayed on the Test Set, make the following settings in the ProComm terminal emulator on the PC:

---

### NOTE

The ProComm terminal emulator views the file transfer as sending the file from the PC “up” to the Test Set. This is opposite to the direction used by the previous Windows Terminal example. Therefore, with ProComm an ASCII “upload” transfer is used.

---

1. Press Alt+F10 to display the ProComm help screen.
2. Press Alt+P to display the SETUP MENU.
3. Select item 6: ASCII TRANSFER SETUP.
4. Set Echo locally: NO.
5. Expand blank lines: YES.
6. Pace character: 0.
7. Character pacing: 15.
8. Line pacing: 10.
9. CR translation: NONE , LF.
10. Translation: NONE (This is important since the default setting will strip line feeds and this will cause the transfer to never begin).
11. Select the Escape key to exit setup mode and return to the main screen.
12. Press Alt F10 to access the help menu.
13. To begin sending the file, select PgUp.
14. In the UPLOAD screen, select 7 ASCII protocol.
15. Run the ASCII\_DN download program on the Test Set.
16. When the Test Set displays `Ready to receive ASCII file data`, press Enter on the PC to begin the transfer. At this point, each line of the program will rapidly scroll across the screen of the Test Set. When the transfer is finished, the download program will display `Down load of ASCII file complete.`, and the program file will be stored on the SRAM card in the TEMP-CODE file.

17. Before running the transferred program, execute a SCRATCH command on the IBASIC Command Line line to remove the ASCII\_DN download program from Test Set memory.
18. Next, execute a GET TEMP\_CODE command on the IBASIC Command Line. This will load the ASCII text into the IBASIC program memory.
19. Finally, execute a RUN command on the IBASIC Command Line. This will run the program. If any syntax errors are present in the program IBASIC will generate the appropriate error messages.



## Uploading Programs from the Test Set to a PC

As an overview, the following steps must be performed:

1. The Test Set must output the program over Serial Port 9.
2. The PC must receive the data through its serial port and direct the data to a file on disk. This can be done by a terminal emulator program such as Windows Terminal, ProComm, or HP AdvanceLink. This requires having the serial port connection established as outlined in "[Interfacing to the IBASIC Controller using Serial Ports](#)" on page 380.

To configure the Test Set to output the program to Serial Port 9 position the cursor on the IBASIC Command Line field. Execute the command `PRINTER IS 9`. This command sets Serial Port 9 as the default printer port. When `PRINT` commands are executed, ASCII characters will be sent to Serial Port 9.

On the PC, select `Receive Text File` in Windows Terminal or `Receive Files (PgDn which is called Download)` in ProComm. Enter a file name, then initiate the file transfer. The PC is now looking for ASCII text to come in the serial port.

Load the program to be transferred into the Test Set. Execute the IBASIC `LIST` command on the IBASIC Command Line. The program listing will be sent to Serial Port 9 and be received by the terminal emulator software on the PC. When the listing is finished, terminate the file transfer by selecting `Stop` on Windows or `Escape` on ProComm.

## Serial I/O from IBASIC Programs

There are two serial ports available for I/O (input / output) to peripherals external to the Test Set. To bring data in to the Test Set through the serial port(s) use the IBASIC ENTER command. To send data out, use the OUTPUT command.

### Serial Ports 9 and 10

The Test Set uses a small RJ-11 female connector on the rear panel for connecting to the two serial ports. This connector has six wires, 3 for Serial Port Address 9 and 3 for Serial Port Address 10. For information about serial port configuration, refer to the "[Test Set Serial Port Configuration](#)" on page 380. For connection information, refer to [figure 10-4](#) on page 384.

Before using either port, the RS-232 protocol must be established by setting baud rate, pacing, and the other settings as explained in "[Test Set Serial Port Configuration](#)" on page 380. Functionally, from an I/O perspective, the two serial ports are identical. However, operationally there is one major difference. The Serial Port Address 9 settings are adjustable on the I/O CONFIGURE screen or with IBASIC commands, while the Serial Port 10 settings are adjustable only with IBASIC commands. There is no screen for Serial Port 10 settings. For more information, see [Chapter 5, "HP-IB Command Dictionary"](#) which gives the command syntax for Serial Port 9 and 10.

## Example IBASIC Program Using Serial Port 10

The following program illustrates I/O to both serial ports. The program sends a prompt message to a terminal connected to Serial Port 9 and waits for a response from the user at the terminal. When the response is received from the terminal connected to Serial Port 9, a series of ASCII characters are sent out Serial Port 10.

### Example

```

10 !....ASCII CHARACTER CYCLER.....
30 !....be connected to a terminal at 9600 baud.
40 !....Outputs ASCII characters on Serial Port 10 beginning with
   ASCII
50 !.....character 32 (space) and ending with ASCII character 126
   (~).
60 !.....Characters are output with no CR/LF
70 OUTPUT 9;"When you are ready to send data on port10,press ENTER"
80 OUTPUT 800;"CONF:SPOR:SIN 'IBASIC';BAUD '9600'"
90 !Allows IBASIC to read port 9
100 DIM A$(10)
110  ENTER 9;A$ !Program waits here until CR/LF is received.
120 !.....
130 I=32
140 HILE I<=126
150 OUTPUT 10 USING "K,#";CHR$(I)
160 !Outputs characters all on one line.
170 OUTPUT 10 USING "K,#";CHR$(I)
180 !Outputs characters all on one line.
190 END WHILE
200 OUTPUT 800;"CONF:SPOR:SIN 'Inst'" !Sets port 9 to IBASIC entry
   field.
180 EXECUTE ("CURSOR HOME") !Places cursor at left of IBASIC entry field
190 END

```

## Serial Port 10 Information

Serial Port 10 is sometimes called Serial Port B in Test Set documentation and programs.

The default Serial Port 10 settings are the same as Serial Port 9. They are

1. Serial Baud rate: 9600
2. Parity: None
3. Data Length: 8 Bits
4. Stop Length: 1 Bit
5. Receive and Transmit Pacing: Xon/Xoff
6. Serial in: Not available for Port 10
7. IBASIC and Instrument Echo: Not available for Port 10

There is no Test Set screen that shows Serial Port 10's settings. Therefore, to know Serial Port 10 settings, they must either be set or queried using IBASIC commands.

For example, the following IBASIC program queries the baud rate setting of Serial Port 10:

### Example

```
10 DIM Setting$(20)
20 OUTPUT 800;"CONF:SPB:BAUD?" !Initiates a query.
30 ENTER 800;Setting$
40 DISP Setting$
50 END
```

This program returns a quoted string. If the baud rate is set to 9600, the returned ASCII character string is 9600. Serial Port 10 settings are held in non-volatile memory. They remain unchanged until modified using an IBASIC command.

---

## PROGram Subsystem

### Introduction

The PROGram Subsystem provides a set of commands which allow an external controller to generate and control an IBASIC program within the Test Set. The PROGram Subsystem in the Test Set is a limited implementation of the PROGram Subsystem defined in the Standard Commands for Programmable Instruments (SCPI) Standard. The PROGram Subsystem commands, as implemented in the Test Set, can be used to

- download an IBASIC program from an external controller into the Test Set
- upload an IBASIC program from the Test Set into an external controller
- control an IBASIC program resident in the Test Set from an external controller
- set or query program variables within an IBASIC program which is resident in the Test Set
- execute IBASIC commands in the Test Set's IBASIC Controller from an external controller

### SCPI PROGram Subsystem

The SCPI PROGram Subsystem was designed to support instruments which can store multiple programs in RAM memory at the same time. The SCPI PROGram Subsystem provides commands which allow multiple programs to be named, defined and resident in the instrument at the same time. The Test Set does not support this capability.

For complete information on the SCPI PROGram Subsystem refer to the Standard Commands for Programmable Instruments (SCPI) Standard. If you are not familiar with SCPI, it is recommended that you obtain a copy of the book: *A Beginner's Guide to SCPI* (ISBN 0-201-56350, Addison-Wesley Publishing Company).

## Test Set PROGram Subsystem

The Test Set was designed to store only one IBASIC program in RAM memory at any given time. The PROGram Subsystem commands, as implemented in the Test Set, operate differently than described in the SCPI Standard. In addition, the SCPI PROGram Subsystem commands which were designed to support multiple programs are not supported in the Test Set.

### Supported SCPI Commands

The Test Set supports the following subset of the :SElected SCPI commands.

- :SElected:DEFine
- :SElected:DEFine?
- :SElected:DELEte:ALL
- :SElected:EXECute
- :SElected:NUMBer
- :SElected:NUMBer?
- :SElected:STATe
- :SElected:STATe?
- :SElected:STRing
- :SElected:STRing?
- :SElected:WAIT

### Unsupported SCPI Commands

The Test Set does not support the following SCPI commands.

- :CATalog?
- :SElected:DELEte:SElected
- :SElected:MALLocate
- :SElected:MALLocate?
- :SElected:NAME
- :SElected:NAME?
- :EXPLicit:DEFine
- :EXPLicit:DEFine?
- :EXPLicit:DELEte
- :EXPLicit:EXECute

- :EXPLicit:MALLocate
- :EXPLicit:MALLocate?
- :EXPLicit:NUMBer
- :EXPLicit:NUMBer?
- :EXPLicit:STATe
- :EXPLicit:STATe?
- :EXPLicit:STRing
- :EXPLicit:STRing?
- :EXPLicit:WAIT

---

NOTE

Sending the Test Set any of the unsupported SCPI PROGRAm Subsystem commands can result in unexpected and/or erroneous operation of IBASIC. This may require the Test Set's RAM to be initialized from the SERVICE screen to regain proper IBASIC operation.

---

## PROGram Subsystem Commands

See ":PROGram subsystem" on page 208, for PROGram Subsystem command syntax rules.

### Command Notation

The following notation is used in the command descriptions:

- Letter case (uppercase or lowercase) is used to differentiate between the short form (the uppercase characters) and long form (the whole keyword) of the command.
- The lower case letters in the keyword are optional; they can be deleted and the command will still be understood by the Test Set.
- []  
= Optional keyword; this is the default state, the Test Set will process the command to have the same effect whether the optional keyword is included by the programmer or not.
- <>  
= Specific SCPI-defined parameter types. Refer to the SCPI Standard for definitions of the SCPI-defined parameter types.
- {}  
= One or more parameters that must be included one or more times.
- |  
= Separator for choices for a parameter. Can be read the same as "or."

### Command Descriptions

---

#### NOTE

When a PROGram Subsystem command is sent to the Test Set through HP-IB from an external controller the Test Set is put into REMOTE mode. The Test Set must be put in LOCAL mode to use the front-panel keys or to use the serial ports to input data into the IBASIC Command line.

---

**[:SElected]**. All the commands under this keyword access the IBASIC program currently resident in the Test Set. Note that this keyword is optional in the command syntax.

#### Syntax

PROGram[:SElected]

**:DEFine <program>**. The DEFine command is used to create and download an IBASIC program into the Test Set from an external controller.



To download an IBASIC program, any currently resident IBASIC program must first be deleted using the :DELEte:ALL command. Attempting to download a new IBASIC program while an IBASIC program is currently resident causes  
 IBASIC Error: -282 Illegal program name.

---

NOTE

It is possible for the PROGram Subsystem to think that there is an IBASIC program resident in the Test Set when, in actuality, there is not. This situation would exist for example, if an IBASIC program had been created and downloaded using the :DEFine command and then deleted, from the front panel, using the SCRATCH ALL command from the IBASIC Command line. Under this circumstance IBASIC Error -282 would be generated when another attempt is made to download a program with the PROGram Subsystem. It is recommended that the :DELEte:ALL command always be sent immediately before the :DEFine command.

---

The IBASIC program downloaded into the Test Set must be transferred as IEEE 488.2 Arbitrary Block Program Data. Refer to the IEEE Standard 488.2-1987 for detailed information on this data type. Two syntax forms are provided with the Arbitrary Block Program Data data type: one form if the length of the program is known and another one if it is not.

**Syntax (length of program not known)**

PROGram[:SELEcted]:DEFine <#0><program><NL><END>

The following notation is used in the command description:

- <#0> = IEEE 488.2 Arbitrary Block Program Data header.
- <program> = the IBASIC program sent as 8 bit data bytes.
- <NL> = new line = ASCII line-feed character.
- <END> = IEEE 488.1 END message. This terminates the block transfer and is only sent once with the last byte of the indefinite block data.

### Example BASIC program to download an IBASIC program to Test Set

```
10 OUTPUT 714;"PROG:DEL:ALL"!Delete current program
20 OUTPUT 714;"PROG:DEF #0"!Create program, send header
30 OUTPUT 714;"10 FOR J = 1 TO 10"!1st prog line
40 OUTPUT 714;"20 DISP J"!2nd prog line
50 OUTPUT 714;"30 BEEP"!3rd prog line
60 OUTPUT 714;"40 NEXT J"!4th prog line
70 OUTPUT 714;"50 END"END!Send END message at end of last line
80 END
```

### Syntax (length of program known)

```
PROGram[:SElected]:DEFine <#><of digits in count field>
<count field: of data bytes in program><program data bytes>
```

The following notation is used in the command description:

- The data starts with a header which begins with a "#", followed by a single non-zero digit in the range 1-9 which specifies the number of digits in the following count field, followed by a series of digits in the range of 0-9 which gives the number of data bytes being sent, followed by the number of data bytes specified by the count field.

### Example

```
#16<data byte><data byte><data byte><data byte><data byte><data byte>
>
```

### Example BASIC program to download an IBASIC program to Test Set

```
10 OUTPUT 714;"PROG:DEL:ALL" !Delete current program
20 OUTPUT 714;"PROG:DEF #257" !Create program, send header
30 OUTPUT 714;"10 FOR J = 1 TO 10" !18 characters + CR + LF
40 OUTPUT 714;"20 DISP J" !9 characters + CR + LF
50 OUTPUT 714;"30 BEEP" !7 characters + CR + LF
60 OUTPUT 714;"40 NEXT J" !9 characters + CR + LF
70 OUTPUT 714;"50 END"!6 characters
80 END
```

**:DEFine?.** The :DEFine? query command is used to upload an IBASIC program from the Test Set to an external controller.

The IBASIC program uploaded to the external controller is transferred as IEEE 488.2 Definite Length Arbitrary Block Response Data. The following information describes some of the characteristics of the IEEE 488.2 Definite Length Arbitrary Block Response Data type. Refer to the IEEE Standard 488.2-1987 for detailed information on this data type.

- The data starts with a header which begins with a “#”, followed by a single non-zero digit in the range 1-9 which specifies the number of digits in the following count field, followed by a series of digits in the range of 0-9 which gives the number of data bytes being sent, followed by the number of data bytes specified by the count field.

### Example

```
#16<data byte><data byte><data byte><data byte><data byte><data byte>
```

- The transfer is terminated by the transmission, from the Test Set to the external controller, of the response message terminator (NL & END message).
- <NL> = new line = ASCII linefeed character.
- <END> = IEEE 488.1 END message.

### Syntax

```
PROGram[:SElected]:DEFine?
```

### Example BASIC program to upload an IBASIC program from Test Set

```
10 DIM Prog_line$(200)!Holds longest program line in Test Set
20 DIM File_name$(10)!Holds the name of file to store IBASIC program
30 LINPUT "Enter name of file to store IBASIC program in:",File_name$
40 OUTPUT 714;"PROG:DEF?"
50 ENTER 714 USING "X,D,#";Count_length !Get length of count field
60 !Get number of characters in program, includes CR/LF on each line
70 ENTER 714 USING VAL$(Count_length)&"D,#";Chars_total
80 !Create ASCII file to hold program, add 5 records for buffer
90 CREATE ASCII File_name$, (Chars_total/256)+5
100 ASSIGN @File TO File_name$
110 LOOP
120 ENTER 714;Prog_line$ !Read in one program line
130 OUTPUT @File;Prog_line$ !Store in file
140 Chars_xferd=Chars_xferd+LEN(Prog_line$)+2 !CR/LF not read
150 EXIT IF Chars_xferd>=Chars_total
160 END LOOP
170 ENTER 714;Msg_terminator$ !Terminate the block data transfer
180 ASSIGN @File TO *
190 END
```

**:DELeTe:ALL.** The :DELeTe:ALL command is used to delete an IBASIC program in the Test Set. If the IBASIC program in the Test Set is in the RUN state, an IBASIC Error: -284 Program currently running error is generated and the program is not deleted.

### Syntax

```
PROGram[:SELeCted]:DELeTe:ALL
```

### Example

```
OUTPUT 714;"PROGram:SELeCted:DELeTe:ALL"
```

or

```
OUTPUT 714;"PROG:DEL:ALL"
```

**:EXECute <program\_command>.** The :EXECute command is used to execute, from an external controller, an IBASIC program command in the Test Set's built-in IBASIC Controller.

<program\_command> is string data representing any legal IBASIC command. If the string data does not represent a legal IBASIC command, an IBASIC Error: -285 Program syntax error is generated.

Any IBASIC program in the Test Set must be in either the PAUSed or STOPped state before the external controller issues the :EXECute <program\_command> command. If the IBASIC program is in the RUN state, an IBASIC Error: -284 Program currently running is generated.

### Syntax

```
PROGram[:SELeCted]:EXECute <delimiter><program_command><delimiter>
```

The following notation is used in the command description:

- <delimiter> = IEEE 488.2 <string data> delimiter, single quote or double quote, must be the same.

### Example

```
OUTPUT 714;"PROGram:SELeCted:EXECute 'CLEAR SCREEN' "
```

or

```
OUTPUT 714;"PROG:EXEC 'CLEAR SCREEN' "
```

**:NUMBer <varname>{<nvalues>}** . The :NUMBer command is used to set, from an external controller, the value of numeric variables or arrays in an IBASIC program in the Test Set. <varname> is the name of an existing numeric variable or array, and can be sent as either character data (<varname> not enclosed in quotes) or string data (<varname> enclosed in quotes). <nvalues> is a list of comma-separated <numeric\_values> which are used to set the value of <varname>.

---

**NOTE** If the variable name <var\_name> is longer than 12 characters it must be sent as string data (<var\_name> enclosed in quotes). For example, OUTPUT 714;"PROG:NUMB 'Var\_name',10".

Attempting to send a <var\_name> longer than 12 characters as character data (<var\_name> *not* enclosed in quotes) will generate the following error:

HP-IB Error: -112 Program mnemonic too long.

---

If an attempt is made to set the value of a numeric variable or array and no IBASIC program is in the Test Set an IBASIC Error: -282 Illegal program name is generated. If an attempt is made to set the value of a numeric variable or array and the numeric variable specified in <varname> does not exist in the program an IBASIC Error: -283 Illegal variable name is generated. If the specified numeric variable cannot hold all of the specified <numeric\_values> an IBASIC Error: -108 Parameter not allowed is generated.

### Syntax

PROGram[:SElected]:NUMBer <varname>{,<nvalues>}

### Example setting the value of a simple variable

OUTPUT 714;"PROG:SElected:NUMBer Variable,15"

or

OUTPUT 714;"PROG:NUMB Variable,15"

### Example setting the value of a one dimensional array [Array(5)] with 6 elements

OUTPUT 714;"PROG:SElected:NUMBer Array,0,1,2,3,4,5"

or

OUTPUT 714;"PROG:NUMB Array,0,1,2,3,4,5"

---

**NOTE** Individual array elements cannot be set with the :NUMBER command.

### Example setting the value of a two dimensional array [Array(1,2)] with 6 elements

OUTPUT 714;"PROG:SElected:NUMBer Array,0,1,2,3,4,5"

or

OUTPUT 714;"PROG:NUMB Array,0,1,2,3,4,5"

Arrays are filled by varying the right-most dimension the fastest. After executing the above statement the array values would be, Array(0,0)=0, Array(0,1)=1, Array(0,2)=2, Array(1,0)=3, Array(1,1)=4, Array(1,2)=5.

---

**NOTE** Individual array elements cannot be set with the :NUMBER command.

**:NUMBer? <varname>** . The :NUMBer? query command is used to return, to an external controller, the current value of numeric variables or arrays in an IBASIC program in the Test Set. <varname> is the name of an existing numeric variable or array in the IBASIC program, and can be sent as either character data (name not enclosed in quotes) or string data (name enclosed in quotes).

---

**NOTE**

Attempting to send a <var\_name> longer than 12 characters as character data (<var\_name> *not* enclosed in quotes) will generate the following error:

If the variable name <var\_name> is longer than 12 characters it must be sent as string data (<var\_name> enclosed in quotes). For example, OUTPUT 714;"PROG:NUMB 'Var\_name'".

HP-IB Error: -112 Program mnemonic too long.

---

For simple variables the value is returned as a series of ASCII characters representing a numeric value in scientific notation (+3.0000000000E+000). For arrays the values are returned as a comma separated list of ASCII characters representing a numeric value in scientific notation. For example, +3.0000000000E+000,+3.0000000000E+000,+3.0000000000E+000, etc.

Array values are sent by varying the rightmost dimension of the array the fastest.

If an attempt is made to query the value of a numeric variable or array and no IBASIC program is in the Test Set an IBASIC Error: -283 Illegal variable name is generated. If an attempt is made to query the value of a numeric variable or array and the variable specified in <varname> does not exist in the program an IBASIC Error: -283 Illegal variable name is generated.

**Syntax**

PROGram[:SElected]:NUMBer? <varname>

---

**NOTE**

The program commands and syntax used to enter data from the Test Set into the external controller will depend upon the programming language used in the external controller. Considerations such as type conversion (integer to real, real to complex, etc.), the sequence in which values are entered into arrays, the capability to fill an entire array with a single enter statement, etc. will depend upon the capabilities of the programming language used in the external controller. The examples which follow represent the capabilities of HP Rocky Mountain BASIC programming language running on an HP 9000/300 Series Controller.

---

### Example querying the value of a simple variable

```
OUTPUT 714;"PROG:SElected:NUMBer? Variable"
ENTER 714;Value
```

or

```
OUTPUT 714;"PROG:NUMB? Variable"
ENTER 714;Value
```

This example assumes that the variable named Value in the ENTER statement is the same type as the variable named Variable in the IBASIC program.

### Example querying the value of a one dimensional array [Array(5)] with 6 elements

```
OUTPUT 714;"PROG:SElected:NUMBer? Array"
ENTER 714;Result_array(*)
```

or

```
OUTPUT 714;"PROG:NUMB? Array"
ENTER 714;Result_array(*)
```

This example assumes that the array named Result\_array(\*) in the ENTER statement is dimensioned exactly the same as the array named Array in the IBASIC program.

---

#### NOTE

Individual array elements cannot be queried with the :NUMBer? command.

### Example querying the value of a one dimensional array whose name is known but whose current size is unknown

```
10 DIM Temp$[5000] !This will hold 250 numbers @ 20 characters each
20 DIM Result_array(500) !This array will hold up to 501 values
30 OUTPUT 714;"PROG:NUMB? Array" !Query the desired array
40 ENTER 714;Temp$ !Enter the values into a temporary string variable
50 N=-1 !Initialize array pointer, assume option base 0
60 REPEAT !Start loop to take values from string and put in array
70 N=N+1 !Increment array pointer
80 Pos_comma=POS(Temp$,",") !Find comma separator
90 Result_array(N)=VAL(Temp$[1,Pos_comma-1]) !Put value into array
100 Temp$=Temp$[Pos_comma+1] !Remove value from temporary string
110 UNTIL POS(Temp$,",")=0 !Check for last value in temporary string
120 Result_array(N+1)=VAL(Temp$) !Put last value into array
130 END
```

The above example assumes that the dimensioned size of the IBASIC array is smaller than the dimensioned size of the array named Result\_array.

---

#### NOTE

Individual array elements cannot be queried with the :NUMBer? command.

**:STATe RUN | PAUSE | STOP | CONTinue** . The STATe command is used to set, from an external controller, the execution state of the IBASIC program in the Test Set. [Table 10-5](#) defines the effect of setting the execution state of the IBASIC program to a desired state from each of the possible current states.

**Table 10-5 Effect of STATe Commands**

Desired State of IBASIC Program (STATe command sent to Test Set)	Current State of IBASIC Program		
	RUNNING	PAUSED	STOPPED
RUN	HP-IB Error: -221 Settings conflict	RUNNING	RUNNING
CONT	HP-IB Error: -221 Settings conflict	RUNNING	HP-IB Error: -221 Settings conflict
PAUSE	PAUSED	PAUSED	STOPPED
STOP	STOPPED	STOPPED	STOPPED

The program execution states are defined as follows:

- RUNNING, the program is currently executing.
- PAUSED, the program has reached a break in execution but can be continued.
- STOPPED, program execution has been terminated.

**Syntax**

PROGram[:SELEcted]:STATe RUN|PAUSE|STOP|CONTinue

**Example**

OUTPUT 714: "PROGram:SELEcted:STATe RUN"

or

OUTPUT 714: "PROG:STAT RUN"

**:STATe?**. The STATe? query command is used to query, from an external controller, the current execution state of the IBASIC program in the Test Set. The return data (RUN, STOP, or PAUS) is sent as a series of ASCII characters.

The program execution states are defined as follows:



- RUN, the program is currently executing.
- PAUS, the program has reached a break in execution but can be continued.
- STOP, program execution has been terminated.

### Syntax

```
PROGram[ :SElected]:STATe?
```

### Example

```
OUTPUT 714;"PROGram:SElected:STATe?"
```

```
ENTER 714;State$
```

or

```
OUTPUT 714;"PROG:STAT?"
```

```
ENTER 714;State$
```

**:STRing <varname>{,<svalues>}** . The :STRing command is used to set, from an external controller, the value of string variables or string arrays in an IBASIC program in the Test Set. <varname> is the name of an existing string variable or string array in the IBASIC program. <svalues> is a list of comma-separated quoted strings which are used to set the value of <varname>.

---

#### NOTE

If the variable name <var\_name> is longer than 12 characters it must be sent as string data (<var\_name> enclosed in quotes). For example, OUTPUT 714;"PROG:STR 'Var\_name','data'".

Attempting to send a <var\_name> longer than 12 characters as character data (<var\_name> *not* enclosed in quotes) will generate the following error:

```
HP-IB Error: -112 Program mnemonic too long.
```

---

#### NOTE

If the programmer wishes to append the IBASIC "\$" string identifier onto the string variable name, the string variable name must be sent as string data, that is enclosed in quotes. For example, OUTPUT 714;"PROG:STR 'Var\_name\$','data'".

Appending the IBASIC "\$" string identifier onto the string variable name without enclosing the string variable name in quotes will generate

```
HP-IB Error: -101 Invalid character.
```

If an attempt is made to set the value of a string variable or array and no IBASIC program is in the Test Set an IBASIC Error: -282 Illegal program name is generated. If an attempt is made to set the value of a string variable or array and the string variable specified in <varname> does not exist in the program an IBASIC Error: -283 Illegal variable name is generated. If a quoted string value is too long to fit into the string variable then it is silently truncated when stored into the IBASIC string variable. If the specified string variable cannot hold all of the quoted strings an IBASIC Error: -108 Parameter not allowed is generated.

### Syntax

```
PROGRAM[:SElected]:STRing <varname>{,<svalues>}
```

### Example setting the value of a simple string variable

```
OUTPUT 714;"PROGRAM:SElected:STRing Variable,'data'"
```

or

```
OUTPUT 714;"PROG:STR Variable,'data'"
```

### Example of setting the value of a string array with 3 elements of 5 characters each, such as Array\$(2)[5]

```
OUTPUT 714;"PROGRAM:SElected:STRing Array,'12345','12345','12345'"
```

or

```
OUTPUT 714;"PROG:STR Array,'12345','12345','12345'"
```

---

#### NOTE

With Option Base 0 set in IBASIC, array indexing starts at 0.

**:STRing? <varname>** . The :STRing? query command is used to return, to an external controller, the current value of string variables or arrays in an IBASIC program in the Test Set. <varname> is the name of an existing string variable or string array in the IBASIC program.

---

#### NOTE

If the variable name <var\_name> is longer than 12 characters it must be sent as string data (<var\_name> enclosed in quotes). For example, OUTPUT 714;"PROG:STR? 'Var\_name'".

Attempting to send a <var\_name> longer than 12 characters as character data (<var\_name> *not* enclosed in quotes) will generate the following error:

```
HP-IB Error: -112 Program mnemonic too long
```

---

---

**NOTE**

If the programmer wishes to append the IBASIC '\$' string identifier onto the string variable name, the string variable name must be sent as string data, that is enclosed in quotes. For example,

```
OUTPUT 714;"PROG:STR? 'Var_name$'"
```

Appending the IBASIC '\$' string identifier onto the string variable name without enclosing the string variable name in quotes will generate the following error:

```
HP-IB Error: -101 Invalid character.
```

---

For simple string variables the value is returned as a quoted string ("This is an example."). For string arrays the values are returned as a comma separated list of quoted strings ("This is an example.", "This is an example."). The string array elements are returned in ascending order (Array\$(0), Array\$(1), Array\$(2), etc.).

If an attempt is made to query the value of a string variable or array and no IBASIC program is in the Test Set an IBASIC Error: -283 Illegal variable name is generated. If an attempt is made to query the value of a string variable or array and the string variable specified in <varname> does not exist in the program an IBASIC Error: -283 Illegal variable name is generated.

**Syntax**

```
PROGram[:SElected]:STRing? <varname>
```

---

**NOTE**

The program commands and syntax used to enter string data from the Test Set into the external controller will depend upon the programming language used in the external controller. The examples which follow represent the capabilities of HP Rocky Mountain BASIC programming language running on an HP 9000/300 Series Controller.

---

**Example of querying the value of a simple string variable**

```
OUTPUT 714;"PROGram:SElected:STRing? Variable"
```

```
ENTER 714;Value$
```

or

```
OUTPUT 714;"PROG:STR? Variable"
```

```
ENTER 714;Value$
```

**Example of querying the value of a string array with 3 elements of 5 characters each, such as Array\$(2)[5]**

```
OUTPUT 714;"PROGram:SElected:STRing? Array"
```

```
ENTER 714 USING "3(X,5A,2X)";Result_array$(*)
```

or

```
OUTPUT 714;"PROG:STR? Array"
```

```
ENTER 714 USING "3(X,5A,2X)";Result_array$(*)
```

This example assumes that the string array named Result\_array\$(\*) is dimensioned exactly the same as the array named Array in the IBASIC program and that each element in the string array Array has five characters in it.

### Example of querying the value of a string array whose name is known but whose current size is unknown

```
05 OPTION BASE 1
10 DIM Temp$(5000) !This will hold 5000 characters
20 DIM Temp_array$(50)[200]!Temp array: 50 elements of 200 character
30 OUTPUT 714;"PROG:STR? Array" !Query the desired array
40 ENTER 714;Temp$ !Enter the values into a temporary string variable
50 N=0 !Initialize array pointer
60 EPEAT !Start loop to take values from string and put in array
70 N=N+1 !Increment array pointer
80 Pos_comma=POS(Temp$,",") !Find comma separator
90 Temp_array$(N)=Temp$[2,Pos_comma-2] !Put value into array
100 Temp$=Temp$[Pos_comma+1] !Remove value from temporary string
110 UNTIL POS(Temp$,",")=0 !Check for last value in temporary string
120 Temp_array$(N+1)=Temp$[2,LEN(Temp$)-1]!Put last value in array
130 END
```

The above example assumes that the total number of characters in the dimensioned size of the IBASIC string array named Array is smaller than the dimensioned size of the string variable named Temp\$. Also, the maximum length of any element in the IBASIC string array Array must be less than or equal to 200 characters.

**:WAIT.** The :WAIT command stops the Test Set from executing any commands or queries received through HP-IB until after the IBASIC program exits the RUN state; that is, the program is either PAUSED or STOPPED.

---

#### CAUTION

The Test Set will continue to process HP-IB commands into the HP-IB input buffer up to the point that the buffer is full. If the external controller attempts to send more commands than can fit into the HP-IB input buffer before the IBASIC program is PAUSED or STOPPED, the HP-IB bus will appear to be locked up. This is due to the fact that the HP-IB bus and the external controller will be in a temporary holdoff state while waiting for the HP-IB input buffer to empty.

---

#### CAUTION

If a query command is sent to the Test Set while the IBASIC program is under the influence of a :WAIT command, no data will be put into the Test Set's Output Queue until the IBASIC program is either PAUSED or STOPPED. If the external controller attempts to enter the queried data before the IBASIC program is PAUSED or STOPPED, the HP-IB bus will appear to be locked up. This is due to the fact that the HP-IB bus and the external controller will be in a temporary holdoff state while waiting for the data to be put into the Output queue to satisfy the enter command.

### Syntax

```
PROGram[:SElected]:WAIT
```

### Example

```
OUTPUT 714;"PROGram:SElected:WAIT"
```

or

```
OUTPUT 714;"PROG:WAIT"
```

**:WAIT?**. The :WAIT? query command stops the Test Set from executing any commands or queries received through HP-IB until after the IBASIC program exits the RUN state, that is - the program is either PAUSED or STOPPED. A 1 is returned in response to the :WAIT? query command when the IBASIC program is either stopped or paused.

---

### CAUTION

When the :WAIT? query command is sent to the Test Set the program running on the external controller will hang on the enter or input statement until the IBASIC program is either STOPPED or PAUSED. This is due to the fact that the HP-IB bus and the external controller will be in a temporary holdoff state while waiting for the Test Set to put a 1 into the Output queue to satisfy the :WAIT? query command.

---

### Syntax

```
PROGram[:SElected]:WAIT?
```

### Example

```
OUTPUT 714;"PROGram:SElected:WAIT?"
```

```
ENTER 714;Dummy
```

or

```
OUTPUT 714;"PROG:WAIT?"
```

```
ENTER 714;Dummy
```

Consider the following example where the user wishes to determine, from an external controller, if the IBASIC program running on the Test Set has finished executing. The example programs show how this might be accomplished with and without using the :WAIT? query command.

#### Example BASIC program without using the :WAIT? query command

```
10 OUTPUT 714;"PROG:STAT RUN"  
20 LOOP  
30 OUTPUT 714;"PROG:STAT?"  
40 ENTER 714;State$  
50 EXIT IF State$="STOP" OR State$="PAUS"  
60 END LOOP  
70 DISP "IBASIC program not running."  
80 END
```

#### Example BASIC program using the :WAIT? query command

```
10 OUTPUT 714;"PROG:STAT RUN"  
20 OUTPUT 714;"PROG:WAIT?"  
30 ENTER 714;Dummy !Program will hang here until IBASIC program stops  
40 DISP "IBASIC program not running."  
50 END
```

## Using the EXECute Command

The PROGRAM:EXECute command can be used to list, edit and control IBASIC programs in the Test Set from an external controller. This eliminates having to use the cursor control knob and provides a more efficient way of making small changes to programs. The full range of IBASIC program commands can be executed from an external controller using the PROGRAM:EXECute command.

The following operations are given as typical examples of using the PROGRAM:EXECute command.

---

#### NOTE

The program commands and syntax used to send data from the external controller to the Test Set will depend upon the programming language used in the external controller. The examples which follow represent the capabilities of HP Rocky Mountain BASIC programming language running on an HP 9000/300 Series Controller.

---

#### NOTE

When a PROGRAM Subsystem command is sent to the Test Set through HP-IB from an external controller the Test Set is put into REMOTE mode. The Test Set must be put in LOCAL mode to use the front panel keys or to use the serial ports to input data into the IBASIC Command line.

---

## Entering a new IBASIC program line

IBASIC program lines can be entered directly into the Test Set's RAM memory, one line at a time, from an external controller using the PROGram:EXECute command as follows:

### Example

```
PROG:EXEC '<new program line number/program line>'
```

where <new program line number/program line> represents a valid IBASIC program line.

For example, to enter the following new program line into the Test Set,

### Example

```
20 A=3.14
```

execute the following command from the external controller:

### Example

```
OUTPUT 714;"PROG:EXEC '20 A=3.14'"
```

Quoted strings, such as those used in PRINT commands, must use double quotes. For example,

### Example

```
OUTPUT 714;"PROG:EXEC '30 PRINT ""TEST""'"
```

## Editing an existing IBASIC program line

Existing IBASIC program lines which are resident in the Test Set's RAM memory can be edited, one line at a time, from an external controller using the PROGram:EXECute command as follows:

### Example

```
PROG:EXEC '<existing program line number/modified program line>'
```

where <existing program line number/modified program line> represents an existing IBASIC program line.

For example, to edit the following existing program line in the Test Set.

### Example

```
30 OUTPUT 814;"AFAN:DEMP:GAIN 20 dB"
```

to

```
30 OUTPUT 814;"AFAN:DEMP:GAIN 10 dB"
```

execute the following command from the external controller:

### Example

```
OUTPUT 714;"PROG:EXEC '30 OUTPUT 814;"AFAN:DEMP:GAIN 10 dB""'"
```

Quoted strings, such as those used in OUTPUT commands, must use double quotes.

## Listing A Program

Execute the following command on the external controller to list an IBASIC program which is resident in the Test Set to the currently specified IBASIC Controller LIST device.

### Example

```
OUTPUT 714;"PROG:EXEC 'LIST' "
```

## Downloading An IBASIC Program Into the Test Set

The following procedure uses the PROGram Subsystem commands to transfer an IBASIC program, which is resident in the memory of the external controller, from the external controller to the Test Set. This procedure assumes the Test Set's

HP-IB address is set to 14. The example also assumes the external controller is an HP 9000 Series 300 Controller.

1. Access the Test Set's TESTS (IBASIC Controller) screen.
2. Enter a program into the external controller. Use the sample program below if no program is available. When run, the sample program clears the Test Set's IBASIC Controller display area, and prints a message indicating that the download procedure worked.

### Example

```
10 !THIS IS A SAMPLE PROGRAM  
20 CLEAR SCREEN  
30 PRINT "DOWNLOADING COMPLETED"  
40 END
```

3. Execute the following commands on the external controller:

### Example

```
OUTPUT 714;"PROG:DEL:ALL "  
OUTPUT 714;"PROG:DEF #0 "  
LIST #714  
OUTPUT 714;" "END
```



- To verify that the program was downloaded, execute the following commands from the external controller:

**Example**

```
OUTPUT 714;"PROG:EXEC 'LIST' "
```

The program should be listed on the Test Set's TESTS (IBASIC Controller) screen.

- Run the program on the Test Set by first selecting the LOCAL key on the front panel of the Test Set and then selecting the Run key on the Test Set's TESTS (IBASIC Controller) screen.

**Uploading a Program From the Test Set**

The following BASIC program copies an IBASIC program from the Test Set's IBASIC Controller RAM to the external controller and then stores it to a file on the external controller's currently assigned mass storage device.

When the upload program is entered and run on the external controller, the operator is prompted for the name of the file to store the IBASIC program in. As the upload program is running, the total number of characters in the program, and the number of characters transferred, are displayed.

**Example**

```
10 !Upload an IBASIC program in Test Set to an external controller.
20 DIM Prog_line${200} !Holds longest program line in Test Set
30 DIM File_name${10} !Holds the name of file to store IBASIC program
40 Addr=714 !Test Set HP-IB address
50 LINPUT "Enter name of file to store IBASIC program in:",File_name$
60 OUTPUT Addr;"PROG:DEF?"
70 ENTER Addr USING "X,D,#";Count_length !Get length of count field
80 !Get number of characters in program, includes CR/LF on each line
90 ENTER Addr USING VAL$(Count_length)&"D,#";Chars_total
100 !Create ASCII file to hold program, add 5 records for buffer
110 CREATE ASCII File_name$, (Chars_total/256)+5
120 ASSIGN @File TO File_name$
130 LOOP
140ENTER Addr;Prog_line$ !Read in one program line
150OUTPUT @File;Prog_line$ !Store in file
160Chars_xferd=Chars_xferd+LEN(Prog_line$)+2 !CR/LF not read
170 DISP Chars_xferd;"of";Chars_total;"characters transferred."
180 EXIT IF Chars_xferd>=Chars_total
190 END LOOP
200 ENTER Addr;Msg_terminator$ !Terminate the block data transfer
```

```
210 ASSIGN @File TO * !Close the file  
220 END
```

### **Saving an IBASIC Program To A Memory Card**

The following procedure can be used to save an IBASIC program from the IBASIC Controller's RAM memory to a memory card inserted into the front panel of the Test Set.

1. Press LOCAL, SHIFT, CANCEL on the Test Set to perform an IBASIC reset.
2. If the memory card has not been initialized, insert it into the Test Set and execute the following command on the external controller:

#### **Example**

```
OUTPUT 714;"PROG:EXEC 'INITIALIZE" ":INTERNAL,4" "' "
```

#### **Example**

```
OUTPUT 714;"PROG:EXEC 'INITIALIZE" "DOS:INTERNAL,4" "' "
```

3. Insert the initialized memory card into the Test Set.
4. Define the memory card as the Mass Storage device by executing the following command on the external controller:

#### **Example**

```
OUTPUT 714;"PROG:EXEC 'MSI " ":INTERNAL,4" "' "
```

5. Save the program to the memory card by executing the following command on the external controller:

#### **Example**

```
OUTPUT 714;"PROG:EXEC 'SAVE " "<filename>" "' "
```

---

## The TESTS Subsystem

The Test Set makes available to the user an automated user-interface which has been specifically designed for radio test. One of the primary problems associated with automated radio testing is the need to rapidly configure the software with the information needed to test a specific type of radio. Information such as, test frequencies/channels, test specifications, test parameters, test conditions and pass/fail limits. Most often the test(s) and test procedure(s) used to test a class of radio (AM, FM, AMPS, TACS, TDMA, CDMA, etc.) are defined by an industry standard and are used to test all radio types within that class. However, for a specific radio type, the test(s) may remain the same but the information needed to test the radio changes. For example, a portable hand-held may have different transmit power levels than a mobile - the RF power test is the same but the power levels, supply voltages, pass/fail limits etc. can be different.

There are two approaches which can be used to provide the software with the information needed to test a radio: a) hardcode the information directly into the software, or b) store the information outside the program code itself and make it available to the software as needed. Hardcoding the information into the software has several serious drawbacks: changing the information is difficult and the software becomes specific to that radio type. Storing the information outside the program code and making it available to the software as needed overcomes both of these problems, that is - the information is easy to change and the software is not specific to a particular type of radio.

The Test Set's automated user-interface was designed using this approach. Hewlett-Packard has developed software specifically designed to run on the Test Set. The HP 11807 Radio Test Software provides the user with a library of industry standard tests. All radio specific information has been removed from the software. The information needed to test a specific type of radio is available to the user through the TESTS Subsystem. To generate, change and maintain this radio specific information the TESTS Subsystem provides menu driven input screens to define specifications, parameters, test sequencing and system configuration for a particular radio type.

## Writing Programs For the TESTS Subsystem

The HP 83224A IBASIC Developer's Tool Kit for Windows is required for developing programs which use the Tests Subsystem. Contact your local Hewlett-Packard sales representative or sales office for ordering and pricing information.

## TESTS Subsystem File Descriptions

Three types of files are used in the TESTS Subsystem to store different types of information.

### Code Files

The first aspect of an automated definition is the code itself. This is just a standard IBASIC Code file that can reside either on the Memory card, on an external disk drive connected to the HP-IB port of the Test Set, or in an internal RAM disk. The name of this file is preceded by a lower case c in the Test Set. This tells the TESTS Subsystem that this particular file contains program code.

The first aspect of an automated definition is the code itself. This is just a standard IBASIC Code file that can reside either on the Memory card, on an external disk drive connected to the HP-IB port of the Test Set, or in an internal RAM disk. The name of this file is preceded by a .PGM file extension in the Test Set. This tells the TESTS Subsystem that this particular file contains program code.

The first aspect of an automated definition is the code itself. This is just a standard IBASIC Code file that can reside either on the Memory card, on an external disk drive connected to the HP-IB port of the Test Set, or in an internal RAM disk. The name of this file is preceded by a lower case c in the HP 8921A. This tells the TESTS Subsystem that this particular file contains program code.

### Library Files

A Library indicates all of the available test subroutines in the code, the set of all parameters that might be entered using the user-interface screens, and all specifications that might be used by the subroutines in the code to decide if a test point passes or fails.

Only one Library is defined for each Code file. The name of this file is preceded by a lower case l in the Test Set, telling the TESTS system that this is a Library file. Also, both the Library and Code file should have the same base name to indicate the relationship between them.

Only one Library is defined for each Code file. The name of this file is preceded by a .LIB file extension in the Test Set, telling the TESTS system that this is a Library file. Also, both the Library and Code file should have the same base name to indicate the relationship between them.

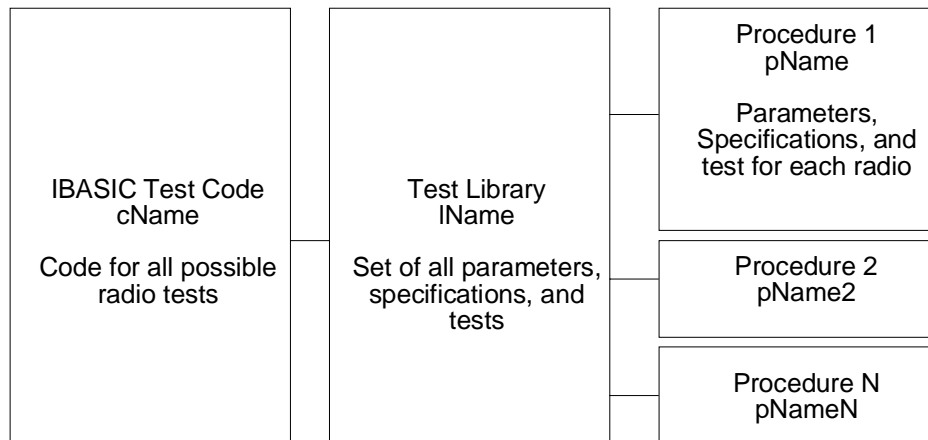
Only one Library is defined for each Code file. The name of this file is preceded by a lower case l in the HP 8921A, telling the TESTS system that this is a Library file. Also, both the Library and Code file should have the same base name to indicate the relationship between them.

A Library is required to use the user-interface screen functions of the TESTS Subsystem. If the program is simple enough that there is no need for user-input, or if all the user-input is simple enough to be accomplished with INPUT statements, a [NO LIB] option is available.

### Procedure Files

A Procedure allows the user to define which of the test subroutines, parameters, and specifications defined in the Library will be used to test a specific Radio. There may be many Procedures defined that use the same IBASIC Code and Library, each using a different subset of the choices available in the Library. These files are preceded with a lower case p in the Test Set.PRC file extent in the Test Set, but are *not* required to have the same base name as either the Library or the Code. The name of the corresponding Library (if any) is stored in each Procedure file.

**Figure 10-7 TESTS Subsystem File Relationship**



ch6drw06.drw

## TESTS Subsystem Screens

The TESTS Subsystem uses several screens to create, select, and copy files, and to run tests.

### The Main TESTS Subsystem Screen

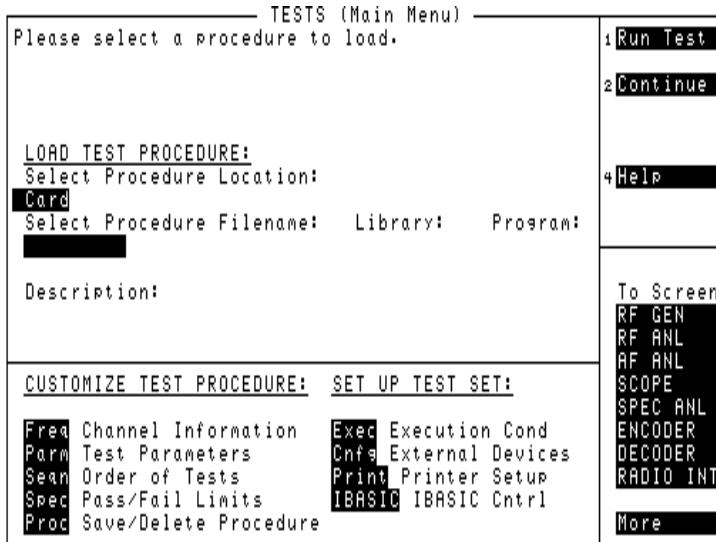
Refer to [figure 10-8](#).

The TESTS (Main Menu) screen is accessed by pressing the front panel TESTS key. Test procedures are selected and run from this screen. Additionally, access to all other TESTS Subsystem screens is accomplished from this screen.

The `Select Procedure Location:` field is used to select the mass storage location for the procedure to be loaded. The `Select Procedure Filename:` field is used to select the name of the procedure to be loaded. The `Description:` field gives the user a brief description of the procedure currently selected in the `Select Procedure Filename:` field.

To view all the Procedures available on the mass storage location currently selected in the `Select Procedure Location:` field, position the cursor on the `Select Procedure Filename:` field and push the rotary knob. A menu will appear in the lower right corner of the screen, displaying all the procedure files which are available. This is not a listing of the full contents of the selected mass storage location, it is only a list of the procedures files that are stored on that media.

**Figure 10-8** The TESTS (Main Menu) Subsystem Screen



ch6dhw7.drw

## TESTS Subsystem User-Interface Screens

The TESTS Subsystem allows the user to easily modify the test subroutines, parameters, specifications and configuration to correspond to the requirements of a specific radio. There are several user-interface screens provided to allow the user to make modifications.

To access any of these screens, position the cursor on the desired field and push the rotary knob.

- The *Order of Tests* screen lets the user select the desired test(s) from the full set of available tests in the loaded procedure file.
- The *Channel Information* screen defines the transmit and receive frequencies used for the selected tests.
- The *Pass/Fail Limits* screen defines the specifications used to generate pass/fail messages during testing.
- The *Test Parameters* screen is used to define instrument settings and characteristics to match those of the radio being tested (audio load impedance, audio power, power supply voltage).
- The *External Devices* screen identifies all connected HP-IB equipped instruments and their HP-IB addresses.
- The *Save/Delete Procedure* screen is used to save or delete Procedures.
- The *Printer Setup* screen is used to select the printer used for IBASIC PRINT commands and to configure the format of the printer page.
- The *Execution Cond* screen is used to configure the IBASIC program execution conditions.
- The *IBASIC Cntrl* screen is the IBASIC Controllers display screen.

Refer to the TESTS screen descriptions in the *Test Set User's Guide* for information concerning how the different TESTS Subsystem screens are used.

The use of the *IBASIC Controller* screen is described in the beginning of this chapter.





---

## **11**      **Error Messages**

## General Information About Error Messages

Information concerning error messages displayed by the Test Set may be found in one of the following manuals:

- *HP 8935 Programmer's Guide*
- *HP 8935 Assembly Level Repair Manual*
- *HP Instrument BASIC User's Handbook:*
- *A Beginner's Guide to SCPI*

The format of the displayed message determines which manual contains information about the error message. There are four basic error message formats:

- Positive numbered error messages
- IBASIC error messages
- HP-IB error messages
- Text only error messages

The following paragraphs give a brief description of each message format and direct you to the manual to look in for information about error messages displayed in that format.

### Positive Numbered Error Messages

Positive numbered error messages are generally associated with IBASIC. Refer to the *HP Instrument BASIC User's Handbook* for information on IBASIC error messages.

Positive numbered error messages take the form:

ERROR XX

For example:

Error 54 Duplicate file name

or

Error 80 in 632 Medium changed or not in drive

## Negative Numbered Error Messages

Negative numbers preceding the error messages text correspond to the error conditions outlined in the Standard Commands for Programmable Instruments (SCPI). For more information on SCPI, order the following book,

*A Beginner's Guide to SCPI* Addison-Wesley Publishing Company ISBN 0-201-56350-9 HP P/N 5010-7166

or contact,

Fred Bode, Executive Director SCPI Consortium

8380 Hercules Drive, Suite P3

La Mesa, CA 91942

Phone: (619) 697-8790, FAX: (619) 697-5955 CompuServe Number:  
76516,254

Negative numbered error messages take the form: ERROR -XX <error message>

For example

```
Error -128 Numeric data not allowed
```

or

```
Error -141 Invalid character data
```

## IBASIC Error Messages

IBASIC Error Messages are associated with IBASIC operation. IBASIC error messages can have both positive and negative numbers. Refer to the *HP Instrument BASIC User's Handbook* for information on positive numbered error messages. Refer to the HP-IB Error Messages section of the *Programmer's Guide* for information on negative numbered error messages (the error message associated with a negative number is the same for HP-IB errors and IBASIC errors).

IBASIC error messages take the form:

```
IBASIC Error: -XX
```

For example:

```
IBASIC Error: -286 Program runtime error
```

## HP-IB Error Messages

HP-IB Error Messages are associated with HP-IB operation. Refer to the Programmers Guide for information on HP-IB error messages.

HP-IB error messages take the form:

```
HP-IB Error: -XX
```

or

```
HP-IB Error
```

For example:

```
HP-IB Error: -410 Query INTERRUPTED.
```

or

```
HP-IB Error: Input value out of range.
```

## Text Only Error Messages

Text only error messages are generally associated with manual operation of the Test Set. This manual contains information on text only error messages.

Text only error messages can also be displayed while running the Test Set's built-in diagnostic or calibration utility programs. Refer to the *Assembly Level Repair* manual for information on text only error messages displayed while running the Test Set's built-in diagnostic or calibration utility programs.

Text only error messages take the form:

```
This is an error message.
```

### Example

```
Input value out of range.
```

## The Message Display

During instrument operation, various messages may appear on the Test Set's display. Prompt-type messages generally appear on the first line of the Test Set's display. General operating and error messages usually appear on the second line of the display. Some messages are persistent; they remain displayed until the error condition no longer exists, or until another persistent message with greater priority occurs. Other messages are only displayed when the error first occurs; they are removed when a key is pressed or the knob is turned, or when an HP-IB command is received. Many of the messages are displayed on the ERROR MESSAGE screen until the instrument is turned off.

Messages that are about error conditions may tell you what to do to correct the error (turn something off, reduce a field's value, press a certain key, and so forth). Messages and prompts are sometimes accompanied by a beep or warble.

---

NOTE

### Warbles and Beeps

A warble sound indicates that an instrument-damaging event is occurring. Beeps often occur only with the first occurrence of the message. Prompts are generally silent.

---

## Non-Recoverable Firmware Error

The non-recoverable firmware error is very important. It appears when an unanticipated event occurs that the Test Set's firmware cannot handle. The message appears in the center of the Test Set's display and (except for the two lines in the second paragraph) has the form:

### Example

```
Non-recoverable firmware error. Please record the 2 lines of
text below and contact Hewlett-Packard through your local
service center or by calling (800) 827-3848 (USA, collect) and
asking to speak to the Service Engineer.
```

```
'Address error exception'
  at line number 0
```

To continue operation, turn POWER off and back on.

Follow the instructions in the message.

Unfortunately, you will not be able to recover from this condition. You must switch the Test Set off and back on. When you rerun the test where the Error Message occurred, it may not occur again. If it does reappear, it would be helpful to Hewlett-Packard to record exactly what the configuration of the instrument was when the error appeared and contact Hewlett-Packard.

## Text Only Error Messages

Operation errors generally occur when you try to do something the Test Set was not designed to do. Most messages tell you what to do to correct the problem, (turn something off, reduce a field's value, press a certain key,...and so forth).

Some common messages are listed here:

- All self tests passed.
- The Test Set did not detect any hardware or firmware failures during its initial self-diagnostics.

This message should always be displayed immediately after instrument turn on.
- Input value out of range.

A number was entered that was too large or small for the selected field, for example, trying to set `AFG1 Freq` to 125 kHz.
- Invalid keystroke.

You used a key that has no function relating to the selected field, for example, pressing the `On/Off` key while the `Filter 1` field is selected.
- Option not installed.

You selected a function that requires optional hardware that is not present.
- Turn off either AM or FM settings.

You tried to create simultaneous AM and FM (using any combination of `AFGen1`, `AFGen2`, and the `Mod In To` field). The Test Set does not provide simultaneous AM and FM.
- Squelch interrupt overflow. Reset using `Meas Reset` (press and release the `Shift` key, then the `Hold` key).

The Test Set temporarily interrupts audio measurements when squelch is first broken to prevent internal switching transients from influencing measurements (except when using the `SCOPE`, `SPEC ANL`, `SERVICE` screens). If squelch is repetitively broken in a period of a few seconds, the duration of measurement interruption becomes too great, and the Test Set stops interrupting the signal. Following measurements may be influenced by transient signals.

`Meas Reset` clears the data buffer used to generate interrupts, resetting the normal squelch operation to eliminate transients.

This condition may occur when monitoring low-level off-the-air signals.

- Cal file checksum incorrect - initializing file.

This error usually occurs after changing the Test Set's firmware ROM's. It is not a problem in that instance, but should not re-appear during subsequent operation of the Test Set.

- One or more self tests failed. Error code: XXXX

An instrument failure was detected when the Test Set was turned on. (For example, having a stuck front-panel key during turn on.) The numbered error message corresponds to a binary-weighted group of errors listed in the \*TST Common Command description in the Programmer's Guide.

- Change Ref Level, Input Port or Attenuator (if using Hold).

The RF signal level is either too great or too small for the current input port, attenuator setting, or both. This error often occurs when trying to make a low-level measurement using the RF IN/OUT port with the spectrum analyzer. Make the indicated changes until this message is no longer displayed.

- Change RF Gen Amplitude, Output Port or Atten Hold (if on).

This message appears when the RF Generator's Amplitude field is set too high when using the RF IN/OUT port or when adjusting the amplitude with the Atten Hold field set to On.

- The RF IN/OUT port has a lower maximum output level than the DUPLEX OUT port. Use the DUPLEX OUT port, or reduce the RF generator's level.

If Atten Hold is On, you may be adjusting the amplitude outside of the allowed range. Change the amplitude

- Direct latch write occurred. Cycle power when done servicing.

- The SERVICE screen was accessed and one or more internal latch settings were changed. Turn the instrument off and back on to reset the latches. (This condition can occur during periodic calibration.)

## **HP-IB Errors**

Most HP-IB errors occur when the control program attempts to query a measurement that is not currently available, or tries to access an instrument connected to the external HP-IB without configuring the Test Set as the System Controller. When diagnosing the cause of an error condition check for these conditions first.



---

## Text Only HP-IB Errors

Un-numbered (text only) HP-IB error messages are generally self-explanatory. For example, trying to retrieve a saved register that does not exist generates the following error message:

```
HP-IB Error: Register does not exist.
```

The following list contains a subset of the Test Set's text only HP-IB error messages. These messages represent error conditions which may require explanation in addition to the error message text.

```
HP-IB Error during Procedure catalog. Check Config.
```

This error occurs when the Test Set fails to access an external HP-IB disk drive when trying to obtain a catalog of procedure files. This would occur when the `Select Procedure Location:` field on the TESTS (Main Menu) screen is set to `Disk` and the operator then tries to select a procedure filename using the `Select Procedure Filename:` field. Ensure that the `Mode` field on the I/O CONFIGURE screen is set to `Control` and that the `External Disk Specification` field on the TESTS (External Devices) screen has the correct mass storage volume specifier for the external disk drive.

```
HP-IB Query Error. Check instrument state.
```

This message usually appears when the control program queries a measurement that is not currently available (on the currently displayed screen and in the ON state), such as querying the TX Frequency measurement when `TX Freq Error` is displayed.

This message may also be immediately followed by the message,

```
HP-IB Error: -420: Query UNTERMINATED.
```

```
HP-IB Error: Not Enough Memory Available for Save.
```

This message will be generated when the control program tries to save the current Test Set state into a Save/Recall register using the `REG:SAVE` commands, but there is insufficient memory available in the Test Set. The Test Set's non-volatile RAM is shared by the following resources:

- IBASIC programs
- Save/Recall registers
- RAM Disk

In order to save the current Test Set state into a Save/Recall register more non-volatile RAM will have to be made available. This can be done by,

- reducing the size of the IBASIC program
- deleting one or more existing Save/Recall registers

- recovering RAM Disk space

The ROM Disk utility RAM\_USAGE will display the total amount of non-volatile RAM installed in the Test Set, the RAM Disk allocation, the Save/Recall register allocation and the amount of non-volatile RAM available to IBASIC.

HP-IB Error: Unknown Save/Recall error.

This error can occur on a Test Set when trying to SAVE the instrument state to a mass storage device with a LIF formatted media. The default file system for the Test Set is DOS. Refer to "[Default File System](#)" on [page 346](#) for more details.

HP-IB Error: HP-IB Units cause invalid conversion of attr.

This error is generated when trying to change Attribute Units and one of the Data Function values is set to zero. If this error is encountered the programmer must change the Data Function settings to values that can be converted to the new units\_of\_measure. Refer to "[Attribute Units \(AUNits\)](#)" on [page 70](#) for more details.

---

## Numbered HP-IB Error Descriptions

The following HP-IB errors can be generated under any of the following conditions:

- controlling the Test Set with an IBASIC program running on the built-in IBASIC controller
- controlling HP-IB devices/instruments, connected to the Test Set's external HP-IB bus, with an IBASIC program running on the built-in IBASIC controller
- controlling the Test Set with a program running on an external controller
- using the Test Set manually to print to an external HP-IB printer
- using the Test Set manually to access procedure/library/code files stored on an external HP-IB disk

---

### NOTE

HP-IB Parser.

The term "Parser" is used in the following error message descriptions. It refers to the Test Set's HP-IB command parser.

---

#### Error -100

Command error

This code indicates only that a Command Error as defined in *IEEE 488.2, 11.5.1.1.4* has occurred.

#### Error -101

Invalid character

A syntactic element contains a character which is invalid for that type.

#### Error -102

Syntax error

An unrecognized command or data type was encountered; for example, a string value was received when the *device* does not accept strings.

#### Error -103

Invalid separator

The parser was expecting a separator and encountered an illegal character. For example, the colon used to separate the **FREQ** and **AMPL** commands should be omitted in the following command:

```
RFG:FREQ 850 MHZ
```

```
:
```

;AMPL -35

**Error -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.

**Error -105**

GET not allowed

A Group Execute Trigger was received within a program message (see *IEEE 488.2, 7.7*).

**Error -108**

Parameter not allowed

More parameters were received than expected for the header. For example, the \*ESE common command only accepts one parameter; receiving \*ESE 36,1 is not allowed.

**Error -109**

Missing parameter

Fewer parameters were received than required for the header. For example, the \*ESE common command requires one parameter; receiving \*ESE is not allowed.

**Error -110**

Command header error

An error was detected in the header.

**Error -111**

Header separator error

A character which is not a legal header separator was encountered while parsing the header.

**Error -112**

Program mnemonic too long

The header contains more than twelve characters (see *IEEE 488.2, 7.6.1.4*).

**Error -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*.

**Error –114**

Header suffix out of range

Indicates that a nonheader character has been encountered in what the parser expects is a header element.

**Error –120**

Numeric data error

This error, as well as errors –121 through –128, are generated when parsing a data element which appears to be numeric, including the nondecimal numeric types.

**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.

**Error –123**

Exponent too large

The magnitude of the exponent was larger than 32000 (see *IEEE 488.2, 7.7.2.4.1*).

**Error –124**

Too many digits

The mantissa of a decimal numeric data element contained more than 255 digits excluding leading zeros (see *IEEE 488.2, 7.7.2.4.1*).

**Error –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.

**Error –130**

Suffix error

This error, as well as errors –131 through –138, are generated when parsing a suffix.

**Error –131**

Invalid suffix

The suffix does not follow the syntax described in *IEEE 488.2 7.7.3.2*, or the suffix is inappropriate for this *device*.

**Error –134**

Suffix too long

The suffix contained more than 12 characters (see *IEEE 488.2, 7.7.3.4*).

**Error –138**

Suffix not allowed

A suffix was encountered after a numeric element which does not allow suffixes.

**Error –140**

Character data error

This error, as well as errors –141 through –148, are generated when parsing a character data element.

**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.

**Error –144**

Character data too long

The character data element contains more than twelve characters (see *IEEE 488.2, 7.7.1.4*).

**Error –148**

Character data not allowed

A legal character data element was encountered where prohibited by the *device*.

**Error –150**

String data error

This error, as well as errors –151 through –158, are generated when parsing a string element.

**Error –151**

Invalid string data

A string data element was expected, but was invalid for some reason (see *IEEE 488.2, 7.7.5.2*). For example, an END message was received before the terminal quote character.

**Error –152**

Parity error

Parity error

**Error –158**

String data not allowed

A string data element was encountered but was not allowed by the *device* at this point in parsing.

**Error -160**

Block data error

This error, as well as errors -161 through -168, are generated when parsing a block data element.

**Error -161**

Invalid block data

A block data element was expected, but was invalid for some reason (see *IEEE 488.2 7.7.6.2*). For example, an END message was received before the length was satisfied.

**Error -168**

Block data not allowed

A legal block data element was encountered but was not allowed by the *device* at this point in parsing.

**Error -170**

Expression error

This error, as well as errors -171 through -178, are generated when parsing an expression data element.

**Error -171**

Invalid expression

The expression data element was invalid (see *IEEE 488.2, 7.7.7.2*); for example, unmatched parentheses or an illegal character.

**Error -178**

Expression data not allowed

A legal expression data was encountered but was not allowed by the *device* at this point in parsing.

**Error -180**

Macro error

This error, as well as errors -181 through -184, are generated when defining a macro or executing a macro.

**Error -181**

Invalid outside macro definition

Indicates that a macro parameter placeholder was encountered outside of a macro definition.

**Error -183**

Invalid inside macro definition

Indicates that the program message unit sequence, sent with a \*DDT or \*DMC command, is syntactically invalid (see .

**Error –184**

Macro parameter error

Indicates that a command inside the macro definition had the wrong number or type of parameters.

**Error –200**

Execution error

This code indicates only that an Execution Error as defined in *IEEE 488.2, 11.5.1.1.5* has occurred.

**Error –201**

Invalid while in local

Indicates that a command is not executable while the *device* is in local due to a hard local control (see *IEEE 488.2, 5.6.1.5*). 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.

**Error –202**

Settings lost due to rtl

Indicates that a setting associated with a hard local control (see *IEEE 488.2, 5.6.1.5*) was lost when the *device* changed to LOCS from REMS or to LWLS from RWLS.

**Error –210**

Trigger error

**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.

**Error –212**

Arm ignored

Indicates that an arming signal was received and recognized by the *device* but was ignored.

**Error –213**

Init ignored

Indicates that a request for a measurement initiation was ignored as another measurement was already in progress.



**Error -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.

**Error -215**

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.

**Error -220**

Parameter error

Indicates that a program data element related error occurred.

**Error -221**

Settings conflict

Indicates that a legal program data element was parsed but could not be executed due to the current device state (see *IEEE 488.2, 6.4.5.3 and 11.5.1.1.5*).

**Error -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* (see *IEEE 488.2, 11.5.1.1.5*).

**Error -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.

**Error -224**

Illegal parameter value

Used where exact value, from a list of possibles, was expected.

**Error -230**

Data corrupt or stale

Possibly invalid data; new reading started but not completed since last access.

**Error –231**

Data questionable

Indicates that measurement accuracy is suspect.

**Error –240**

Hardware error

Indicates that a legal program command or query could not be executed because of a hardware problem in the *device*.

**Error –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.

**Error –250**

Mass storage error

Indicates that a mass storage error occurred.

**Error –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.

**Error –252**

Missing media

Indicates that a legal program command or query could not be executed because of a missing media. For example, no disk.

**Error –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.

**Error –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.

**Error –255**

Directory full

Indicates that a legal program command or query could not be executed because the media directory was full.

**Error -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.

**Error -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.

**Error -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 switch on a memory card was set.

**Error -260**

Expression error

Indicates that an expression program data element related error occurred.

**Error -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.

**Error -270**

Macro error

Indicates that a macro-related execution error occurred.

**Error -271**

Macro syntax error

Indicates that a syntactically legal macro program data sequence, according to *IEEE 488.2, 10.7.2*, could not be executed due to a syntax error within the macro definition (see *IEEE 488.2, 10.7.6.3*).

**Error -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 (see *IEEE 488.2, 10.7.6.3*).

**Error –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* (see *IEEE 488.2, 10.7.3 and 10.7.6.2*). For example, the label was too long, the same as a common command header, or contained invalid header syntax.

**Error –274**

Macro parameter error

Indicates that the macro definition improperly used a macro parameter placeholder (see *IEEE 488.2, 10.7.3*).

**Error –275**

Macro definition too long

Indicates that a syntactically legal macro program data sequence could not be executed because the string of block contents were too long for the device to handle (see *IEEE 488.2, 10.7.6.1*).

**Error –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 (see *IEEE 488.2 10.7.6.6*).

**Error –277**

Macro redefinition not allowed

Indicates that syntactically legal macro label in the *\*DMC* command could not be executed because the macro label was already defined (see *IEEE 488.2, 10.7.6.4*).

**Error –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.

**Error –280**

Program error

Indicates that a downloaded program-related execution error occurred.

**Error –281**

Cannot create program

Indicates that an attempt to create a program was unsuccessful. A reason for the failure might include not enough memory.

**Error -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.

**Error -283**

Illegal variable name

An attempt was made to reference a nonexistent variable in a program.

**Error -284**

Program currently running

Certain operations dealing with programs are illegal while the program is running. For example, deleting a running program is not possible.

**Error -285**

Program syntax error

Indicates that syntax error appears in a downloaded program.

**Error -286**

Program runtime error

**Error -300**

Device-specific error

This code indicates only that a Device-Dependent Error as defined in *IEEE 488.2, 11.5.1.1.6* has occurred.

**Error -310**

System error

Indicates that some error, termed “system error” by the device, has occurred.

**Error -311**

Memory error

Indicates that an error was detected in the *device's* memory.

**Error -312**

PUD memory lost

Indicates that the protected user data saved by the \*PUD command has been lost.

**Error -313**

Calibration memory lost

Indicates that nonvolatile calibration data used by the \*CAL? command has been lost.

**Error –314**

Save/recall memory lost

Indicates that the nonvolatile data saved by the \*SAV command has been lost.

**Error –315**

Configuration memory lost

Indicates that nonvolatile configuration data saved by the *device* has been lost.

**Error –330**

Self-test failed

**Error –350**

Queue overflow

This code indicates that there is no room in the queue and an error occurred but was not recorded. This code is entered into the queue in lieu of the code that caused the error.

**Error –400**

Query error

This code indicates only that a Query Error as defined in *IEEE 488.2 11.5.1.1.7 and 6.3* has occurred.

**Error –410**

Query INTERRUPTED

Indicates that a condition causing an INTERRUPTED Query error occurred (see *IEEE 488.2, 6.3.2.3*). For example, a query followed by DAB or GET before a response was completely sent.

This message appears when you query a measurement without immediately entering the returned value into a variable. For example, the following program lines query the TX Frequency measurement and enter its value into a variable (Rf\_freq):

```
OUTPUT 714;"MEAS:RFR:FREQ:ABS?"  
ENTER 714;Rf_freq
```

**Error –420**

Query UNTERMINATED

Indicates that a condition causing an UNTERMINATED Query error occurred (see *IEEE 488.2, 6.3.2.2*). For example, the *device* was addressed to talk and an incomplete program message was received.

This message usually appears when trying to access a measurement that is not active. For example, you cannot query the DTMF Decoder measurements from the DUPLEX TEST screen, or query the TX Frequency measurement when the TX Freq Error measurement is displayed.

#### Error -430

Query DEADLOCKED

Indicates that a condition causing a DEADLOCKED Query error occurred (see *IEEE 488.2, 6.3.1.7*). For example, both input buffer and output buffer are full and the device cannot continue.

#### Error -440

Query UNTERMINATED after indefinite response

Indicates that a query was received in the same program message after a query requesting an indefinite response was executed (see *IEEE 488.2, 6.5.7.5.7*).

#### Error -606

Update of Input Module Relay Switch Count file failed.

Indicates that the Test Set was not able to update the Input Module Relay Switch Count EEPROM file with the current switch count data from the non-volatile RAM switch count array. This error is most probably generated as a result of a hardware error or failure. Refer to the Test Set's *Assembly Level Repair* for diagnostic information.

#### Error -607

Checksum of Non-Volatile RAM Relay Count data bad.

Indicates that the Test Set was not able to generate the proper checksum for the Input Module Relay Switch Count data from the non-volatile RAM switch count array. This error is most probably generated as a result of a hardware error or failure. Refer to the Test Set's for diagnostic information.

#### Error -608

Initialization of Input Module Relay Count file failed.

Indicates that the Test Set was not able to initialize the Input Module Relay Switch Count EEPROM file during installation of a new input module. This error is most probably generated as a result of a hardware error or failure. Refer to the Test Set's for diagnostic information.

#### Error -1300

Order attempted while not in Connect state.

Indicates that an attempt was made to send an order type Mobile Station Control Message (that is - order a change in power level, put the mobile station in maintenance mode, or send an alert message to the mobile station) when the Call Processing Subsystem was not in the Connect state.

**Error -1301**

Handoff attempted while not in Connect state.

Indicates that an attempt was made to handoff a mobile station to a new voice channel when the Call Processing Subsystem was not in the Connect state.

**Error -1302**

Release attempted while not in Connect state.

Indicates that an attempt was made to send a Release message to a mobile station when the Call Processing Subsystem was not in the Connect state.

**Error -1303**

Page attempted while not in Active state.

Indicates that an attempt was made to Page a mobile station when the Call Processing Subsystem was not in the Active state.

**Error -1304**

Origination attempted while not in Active state.

Indicates that a mobile station attempted to originate a call to the simulated Base Station when the Call Processing Subsystem was not in the Active state.

**Error -1305**

Registration attempted while not in Active state.

Indicates that an attempt was made to send a Registration message to a mobile station when the Call Processing Subsystem was not in the Active state.

**Error -1306**

Origination in progress.

Indicates that an attempt was made to; register, page, handoff, release, order a change in power level, put the mobile station in maintenance mode, or send an alert message to the mobile station while an origination was in progress.

**Error -1307**

Timeout occurred while attempting to register Mobile.



Indicates that the simulated Base Station's internal timer expired before receiving a response from the mobile station during a registration attempt. The internal timer is set to 20 seconds when the Register state is entered.

**Error -1308**

Timeout occurred while attempting to page Mobile.

Indicates that the simulated Base Station's internal timer expired before receiving a response from the mobile station during a page attempt. The internal timer is set to 20 seconds when the Page state is entered.

**Error -1309**

Timeout occurred while attempting to access Mobile.

Indicates that the simulated Base Station's internal timer expired before receiving a response from the mobile station during an access attempt. The internal timer is set to 20 seconds when the Access state is entered.

**Error -1310**

Timeout occurred while attempting to alert Mobile.

Indicates that the simulated Base Station's internal timer expired before receiving a response from the mobile station during an alert attempt. The internal timer is set to 20 seconds when the alert order is sent to the mobile station.

**Error -1311**

RF power loss indicates loss of Voice Channel.

When any Call Processing Subsystem screen is displayed (except the ANALOG MEAS screen) and the Call Processing Subsystem is in the Connect state, the host firmware constantly monitors the mobile station's transmitted carrier power. If the power falls below 0.0005 Watts the simulated Base Station will terminate the call and return to the Active state. This error message is displayed if the host firmware has detected that the mobile station's carrier power has fallen below the 0.0005 Watts threshold. The call is dropped and the Call Processing Subsystem returns to the Active state.

**Error -1312**

Data from RVC contains invalid bits in word [1,2,3].

Indicates that the decoded data received on the reverse voice channel contains invalid bits in word 1 and/or word 2 and/or word 3. The raw decoded data is displayed in hexadecimal format in the top right-hand portion of the CALL CONTROL screen. Raw decoded data is only displayed when the CALL CONTROL screen Display field is set to Data.

#### **Error –1313**

Timeout occurred while in Maintenance state.

Indicates that the simulated Base Station's internal timer expired before the mobile station was taken out of the maintenance state. The internal timer is set to 20 seconds when the maintenance order is sent to the mobile station.

#### **Error –1314**

Alert attempted while not in Maintenance or Connect state.

Indicates that an attempt was made to send an Alert order to the mobile station when the Call Processing Subsystem was not in the Maintenance state or Active state.

#### **Error –1315**

Data from RECC contains invalid bits in word [1,2,3].

Indicates that the decoded data received on the reverse control channel contains invalid bits in word 1 and/or word 2 and/or word 3. The raw decoded data is displayed in hexadecimal format in the top right-hand portion of the CALL CONTROL screen. Raw decoded data is only displayed when the CALL CONTROL screen Display field is set to Data.

#### **Error –1316**

Incomplete data received on RECC for word [1,2,3].

Indicates that the decoded data received on the reverse control channel did not contain the proper number of bits in word 1 and/or word 2 and/or word 3. The raw decoded data is displayed in hexadecimal format in the top right-hand portion of the CALL CONTROL screen. Raw decoded data is only displayed when the CALL CONTROL screen Display field is set to Data.

#### **Error –1317**

Incomplete data received on RVC for word [1,2,3].

Indicates that the decoded data received on the reverse voice channel did not contain the proper number of bits in word 1 and/or word 2 and/or word 3. The raw decoded data is displayed in hexadecimal format in the top right-hand portion of the CALL CONTROL screen. Raw decoded data is only displayed when the CALL CONTROL screen Display field is set to Data.

---

## Symbols

'LIB' files, [442](#)  
'PGM' files, [442](#)  
'PRC' files, [443](#)  
'\_' files, [356](#)  
'c' files, [356](#), [442](#)  
'l' files, [356](#), [442](#)  
'n' files, [356](#)  
'p' files, [356](#), [443](#)

## A

Active Controller  
  when capability required, [338](#)  
Active Measurement, [30](#)  
AdvanceLink (HP 68333F Version  
  B.02.00) terminal emulator,  
  [391](#), [405](#)  
Annunciators, [30](#)  
Arming measurements, [262](#)  
ASCII Text Files  
  sending with ProComm  
    Communications Software,  
    [413](#)  
  sending with Windows  
    Terminal, [412](#)  
Attribute Units, [70](#)  
Autoranging  
  affect on measurement speed,  
  [264](#)  
Autotuning  
  affect on measurement speed,  
  [264](#)

## B

Battery  
  memory card, [364](#)  
  part numbers, [364](#)  
  replacing, [364](#)

---

## C

Calibration Status Register  
  Group, [300](#)  
  accessing registers contained in, [303](#)  
  condition register bit assignments, [301](#)  
Call Processing Subsystem  
  error messages, described, [469](#)  
Code files, [354](#), [442](#)  
Common Commands, [244](#)  
  CLS, [252](#)  
  ESE, [253](#)  
  ESE?, [253](#)  
  ESR?, [253](#)  
  IDN, [245](#)  
  OPC, [248](#)  
  OPC?, [250](#)  
  OPT, [246](#)  
  PCB, [253](#)  
  RCL, [254](#)  
  RST, [246](#)  
  SAV, [254](#)  
  SRE, [253](#)  
  SRE?, [253](#)  
  STB?, [253](#)  
  TRG, [253](#), [256](#)  
  TST, [247](#)  
  WAI, [252](#)  
Communicate Status Register  
  Group, [312](#)  
  accessing registers contained in, [314](#)  
  condition register bit assignments, [313](#)  
controller, external, [21](#), [32](#)  
conventions used in his manual, [15](#)  
COPY\_PL, [367](#)  
Copying a volume, [368](#)  
Copying files, [368](#)

## D

data functions  
  querying ON/OFF state, [76](#)  
  turning ON and OFF, [75](#)  
Default file system, [346](#)  
Display  
  locking display screen via HP-IB, [269](#)  
Display Units, [65](#)  
DOS file names, [355](#)  
DOS file system, [355](#)  
  initializing media for, [358](#)  
  restrictions, [360](#)  
Downloading programs to Test Set, [400](#), [438](#)

## E

EPSON card (see Memory card), [345](#), [350](#), [351](#), [362](#)  
Error Message Queue Group, [294](#)  
  accessing the error message queue, [296](#)  
Error messages, [447](#)  
  types of, [447](#)  
error messages, [448](#)  
  operation, [452](#)  
External Automatic Control Mode, [21](#)  
External controller, [18](#), [21](#), [32](#)  
External disk drives, [347](#)  
  initializing media for, [358](#)

---

## F

- File names
  - conflicts, 357
  - recommendations, 358
- File system
  - backing up files, 367
  - copying volume, 368
  - DOS, 355
  - DOS file names, 355
  - file name conflicts, 357
  - file naming recommendations, 358
  - file types, 359
  - initializing media, 358, 371
  - LIF, 355
  - LIF file names, 355
  - naming files, 355
  - storing code files, 359
- File types, 359
- Files
  - backing up, 367
  - copying, 368
  - storing, 359
- Front panel
  - functions not programmable, 35
  - ON/OFF key, 75

## H

- Hardware Status Register #1
  - Group, 309
  - accessing registers contained in, 311
  - condition register bit assignments, 309
- Hardware Status Register #2
  - Group, 304
  - accessing registers contained in, 307
  - condition register bit assignments, 304
- HFS (Hierarchical File System), 355
- Hierarchical File System (HFS), 355
- HP 8920A Memory Card Part Numbers, 362
- HP 8920B Memory Card Part Numbers, 362
- HP-IB
  - Active Controller, 31, 337, 338
  - address - displaying, 37
  - address - factory setting, 37
  - address - setting, 37
  - arming measurements, 262
  - Attribute units - changing, 72
  - Attribute units - definition, 70
  - Attribute units - guidelines, 74
  - Attribute units - querying, 74
  - changing a field setting, 33
  - Common Commands, 244
  - Common Commands CLS, 252
  - Common Commands ESE, 253
  - Common Commands ESE?, 253
  - Common Commands ESR?, 253
  - Common Commands IDN, 245
  - Common Commands OPC, 248
  - Common Commands OPC?, 250
  - Common Commands OPT, 246
  - Common Commands PCB, 253
  - Common Commands RCL, 254
  - Common Commands RST, 246
  - Common Commands SAV, 254
  - Common Commands SRE, 253
  - Common Commands SRE?, 253
  - Common Commands STB?, 253
  - Common Commands TRG, 253, 256
  - Common Commands TST, 247
  - Common Commands WAI, 252
  - configuration, 31
  - display units - changing, 65
  - display units - definition, 65
  - display units - guidelines, 66
  - display units - querying, 66
  - downloading programs to Test Set, 400
  - error messages, 447
  - Errors, 447, 454
  - extended addressing, 37
  - external (select code 7), 21, 32
  - getting started, 25
  - Group Execute Trigger (GET), 256
  - HP-IB units - changing, 69
  - HP-IB units - definition, 68
  - HP-IB units - guidelines, 70
  - HP-IB units - querying, 69
  - Increasing measurement speed, 264
  - Increasing measurement speed (see Increasing Measurement Speed), 264
  - Instrument Initialization (see Instrument Initialization), 326
  - internal (select code 8), 21, 32
  - local lockout, 46
  - Local/Remote Triggering Changes, 259
  - locking the display screen, 269
  - making a simple measurement, 34
  - measurement pacing, 262
  - multiple addressing, 37
  - passing control (see Passing Control), 337
  - PROGram commands (see PROGram Subsystem), 419
  - programming examples, 28, 33, 76
  - programming guidelines, 26
  - querying the lock/unlock state of the display screen, 269
  - reading a field setting, 33
  - Service requests (see Service Requests), 317
  - standards, 25
  - STATe command - definition, 75
  - STATe command - guidelines, 76
  - Status reporting (see Status reporting), 270
  - System Controller, 31, 337, 338
  - topics covered, 25
  - Trigger - aborting, 260
  - Trigger commands, 259
  - Trigger event, 256
  - Trigger modes, 257, 260
  - Trigger modes - affect on measurement speed, 260, 264

- Trigger modes - default settings, 258
- Trigger modes - retriggering, 257, 260
- Trigger modes - settings for fastest measurements, 261
- Trigger modes - settings for most reliable measurements, 261
- Trigger modes - settling, 258, 260
- Triggering measurements, 256
- units of measure, 64
- uploading programs to Test Set, 401
- using, 17
- HP-IB command syntax
  - AUNits, 72
  - AUNits?, 74
  - DUNits, 65
  - DUNits?, 66
  - guidelines, 61
  - SPECial
    - DISPlay, 269
    - DISPlay?, 269
  - STATe, 75
  - STATe?, 76
  - TRIGger
    - ABORt, 260
    - IMMediate, 259
    - MODE, 260
  - UNITs, 69
  - UNITs?, 69
  - use of single quotes, 29
  - use of spaces, 29, 62
  - using colons to separate commands, 62
  - using question mark to query setting/field, 63
  - using quotes for strings, 62
  - using semicolon colon command separator, 63, 268
  - using semicolon to output multiple commands, 63
  - using upper/lower case letters, 61
- I**
- IBASIC
  - avoiding program hangs, 29
  - command line, 375
  - Controller, 18, 20, 21, 32
  - Controller - default mass storage location, 352
  - Controller - interfacing to using serial ports, 380
  - Controller architecture, 20, 21
  - Controller screen, 375
  - COPY command, 368
  - copying files, 368
  - default file system, 346
  - EDIT mode - entering/exiting, 404
  - error messages, 447
  - GET command, 359
  - INITIALIZE command, 358, 371
  - initializing media, 358
  - language, 20
  - LOAD command, 359
  - making a simple measurement, 34
  - Mass Storage Volume Specifier (MSI), 365
  - MSI, 365
  - passing control back using PASS CONTROL, 340
  - program development (see Program Development), 377
  - requesting HP-IB active control, 340
  - running programs, 26
  - SAVE command, 359
  - selecting mass storage devices, 353
  - STORE command, 359
  - storing files, 359
- IEEE 488.1
  - compliance, 35, 36
  - Interface Function Capabilities, 36
  - Passing Control (see Passing Control), 337
  - Remote Interface Message Capabilities, 38
  - SRQ (see Service Requests), 317
- IEEE 488.2
  - Common Commands, 244
  - Common Commands CLS, 252
  - Common Commands ESE, 253, 291
  - Common Commands ESE?, 253, 290
  - Common Commands ESR?, 253, 290
  - Common Commands IDN, 245
  - Common Commands OPC, 248
  - Common Commands OPC?, 250
  - Common Commands OPT, 246
  - Common Commands PCB, 253, 338
  - Common Commands RCL, 254
  - Common Commands RST, 246
  - Common Commands SAV, 254
  - Common Commands SRE, 253, 320
  - Common Commands SRE?, 253, 320
  - Common Commands STB?, 253, 274
  - Common Commands TRG, 253, 256
  - Common Commands TST, 247
  - Common Commands WAI, 252
  - Common CommandsRST, 332
  - compliance, 35, 36
  - Output Queue, 292
  - Overlapped commands, 60
  - Sequential commands, 60
  - Standard Event Status Register, 285
  - Status Byte Register, 272
- in this manual, 15
- Increasing measurement speed, 264
  - autoranging, 264
  - autotuning, 264
  - combining ENTER statements, 268
  - combining OUTPUT statements, 267
  - compound commands, 266
  - measurement setup time, 266
  - screen display time, 268
  - speed of control program, 266
- Index
  - AFAN
    - INP, 102
  - instrument function
    - querying ON/OFF state, 76
    - turning ON and OFF, 75
  - Instrument Initialization, 326
    - Device Clear (DCL) HP-IB Bus Command, 334
    - Front panel PRESET key, 330
    - Interface Clear (IFC) HP-IB Bus Command, 336
    - methods of, 327
    - power on reset, 328
    - RST Common Command, 332

---

Selected Device Clear (SDC)  
  HP-IB Bus Command, [335](#)  
Internal Automatic Control Mode,  
  [20](#), [22](#)

**K**  
keys  
  front panel, [35](#)

**L**  
Library files, [442](#)  
  backing up, [367](#)  
LIF file names, [355](#)  
LIF file system, [355](#)  
  initializing media for, [358](#)  
lock up  
  HP-IB bus, [258](#), [262](#)

## M

Manual Control Mode, [19](#), [22](#)  
Mass Storage Devices, [345](#)  
  accessing, [354](#)  
  default locations, [352](#)  
  EPSON cards, [350](#), [351](#), [362](#)  
  initializing media for, [358](#), [371](#)  
  OTP card, [351](#), [362](#)  
  overview, [347](#)  
  PCMCIA cards, [350](#), [351](#), [362](#)  
  RAM Disk, [349](#), [370](#)  
  ROM card, [351](#), [362](#)  
  ROM Disk, [350](#), [361](#)  
  selecting, [353](#)  
  SRAM card, [350](#), [362](#)  
  write protecting, [363](#)  
Mass storage locations  
  default values, [352](#)  
  selecting, [353](#)  
Mass Storage Volume Specifier,  
  [365](#)  
measurement  
  active, [19](#), [30](#)  
  querying ON/OFF state, [76](#)  
  querying value, [19](#), [63](#)  
  recommended sequence, [26](#)  
  turning ON and OFF, [75](#)  
Measurement speed - increasing  
  (see Increasing Measurement  
  Speed), [264](#)  
Memory Cards, [345](#)  
  address, [365](#)  
  battery (see Battery), [364](#)  
  initializing, [358](#)  
  inserting, [362](#)  
  Mass Storage Volume Specifier,  
    [365](#)  
  OTP cards, [351](#)  
  part numbers, [362](#)  
  removing, [362](#)  
  ROM cards, [351](#)  
  SRAM cards, [350](#)  
  using, [362](#)  
  write-protect switch, [363](#)  
message  
  error, [448](#)  
  operation, [452](#)  
  types of, [448](#)  
messages  
  error, [447](#)  
Microsoft® Windows Terminal  
  terminal emulator, [388](#), [405](#)

## O

Operating Modes  
  external automatic control, [18](#),  
    [21](#)  
  internal automatic control, [18](#),  
    [20](#), [22](#)  
  manual control, [18](#), [19](#), [22](#)  
operation messages, [452](#)  
Operation Status Register Group,  
  [282](#)  
  accessing registers contained in,  
    [283](#)  
  Condition Register bit  
    assignments, [282](#)  
OTP Memory card, [347](#), [351](#)  
Output Queue Group, [292](#)  
  accessing the output queue, [294](#)  
Overlapped Commands, [60](#)

## P

Pacing measurements, [262](#)  
Passing Control, [337](#)  
  example programs, [340](#)  
  passing control back  
    automatically, [339](#)  
  passing control back to another  
    controller, [339](#)  
  passing control back using PASS  
    CONTROL, [340](#)  
  passing control to Test Set, [338](#)  
  requesting control from IBASIC,  
    [340](#)  
PC  
  AdvanceLink (HP 68333F  
    Version B.02.00) terminal  
    emulator, [391](#), [405](#)  
  Microsoft® Windows Terminal  
    terminal emulator, [405](#)  
  ProCommr® Revision 2.4.3  
    terminal emulator, [406](#)  
  Serial Port Configuration, [387](#)  
  Terminal emulator, [387](#)  
PCMCIA card  
  Flash ROM, [366](#)  
  initializing, [366](#)  
  OTP, [366](#)  
  SRAM, [366](#)  
PCMCIA card (see Memory card),  
  [345](#), [350](#), [351](#), [362](#)  
printer  
  connecting to HP-IB, [31](#)  
Procedure files, [354](#), [443](#)  
  backing up, [367](#)  
ProCommr® Revision 2.4.3  
  terminal emulator, [406](#)  
Program Development  
  choosing development method,  
    [394](#)  
  IBASIC, [378](#)  
  Method #1 - Using external  
    computer, [396](#)  
  Method #2 - Using IBASIC  
    EDIT mode, [403](#)  
  Method #3 - Using word  
    processor on PC, [407](#)  
  methods of, [378](#)  
program hangs  
  avoiding, [29](#)  
PROGram Subsystem, [400](#), [419](#)  
  commands, [422](#)  
  executing commands, [436](#)



---

## Q

Questionable Data/Signal Register Group, [296](#)  
accessing registers contained in, [299](#)  
condition register bit assignments, [297](#)

## R

RAM Disk, [347](#), [349](#)  
  initializing, [371](#)  
  using, [370](#)  
RAM\_MNG, [370](#)  
RJ-11 jack, [380](#)  
ROM Disk, [347](#), [350](#)  
  using, [361](#)  
ROM Memory card, [347](#), [351](#)

## S

Save/Recall Registers  
  default mass storage locations, [352](#)  
Sequential Commands, [60](#)  
Serial Port, [380](#)  
  cables/adapters for, [382](#)  
  configuration, [380](#), [385](#), [416](#)  
  input buffer length, [386](#)  
  receive/transmit pacing, [386](#)  
  select code 10, [380](#), [416](#), [418](#)  
  select code 9, [380](#), [385](#), [416](#)  
  serial I/O from IBASIC program, [416](#)  
Service Request Enable Register, [318](#)  
  clearing, [321](#)  
  reading, [320](#)  
  writing, [320](#)  
Service Requests, [317](#)  
  enabling SRQ interrupts, [317](#)  
  procedure for generating, [321](#)  
  Service Request Enable Register (see Service Request EnableRegister), [318](#)  
  setting up SRQ interrupts, [317](#)  
SRAM card  
  initializing, [366](#)  
SRAM Memory card, [347](#), [350](#)  
SRQ (see Service Requests), [317](#)  
Standard Event Status Register Group, [285](#)  
  accessing registers contained in, [289](#)  
  bit assignments, [289](#)  
STATE  
  ON/OFF, [30](#)  
Status Byte Register, [272](#)  
  bit assignments, [273](#), [317](#)  
  clearing, [275](#)  
  reading with serial poll, [274](#)  
  reading with STB Common Command, [274](#)  
  writing, [275](#)  
Status reporting, [270](#)  
  Calibration Status Register Group (see Calibration StatusRegister Group), [300](#)  
  clearing the Status Byte Register, [275](#)  
  Communicate Status Register Group (see Communicate StatusRegister Group), [312](#)  
  Condition register definition, [277](#)  
  Enable register definition, [278](#)

- Error Message Queue Group  
(see Error Message Queue Group), [294](#)
- Event register definition, [278](#)
- Hardware Status Register #1 Group (see Hardware StatusRegister #1 Group), [309](#)
- Hardware Status Register #2 Group (see Hardware StatusRegister #2 Group), [304](#)
- Operation Status Register Group (see Operation StatusRegister Group), [282](#)
- Output Queue Group (see Output Queue Group), [292](#)
- Questionable Data/Signal Register Group (see QuestionableData/Signal Register Group), [296](#)
- reading Status Byte Register with serial poll, [274](#)
- reading Status Byte Register with STB  
CommonCommand, [274](#)
- Standard Event Status Register Group (see Standard EventStatus Register Group), [285](#)
- Status Byte Register, [272](#)
- status queue model, [279](#)
- status register model, [276](#)
- status register structure overview, [276](#)
- status registers in Test Set, [280](#)
- status reporting structure operation, [279](#)
- structure overview, [270](#)
- Summary Message definition, [278](#)
- Transition filter definition, [277](#)
- writing the Status Byte Register, [275](#)
- Storing code files, [359](#)
- System Controller, [338](#)
- T**
- Terminal Configuration, [393](#)
- Test Set
  - Attribute units - changing, [72](#)
  - Attribute units - definition, [70](#)
  - Attribute units - guidelines, [74](#)
  - Attribute units - querying, [74](#)
  - default file system, [346](#)
  - display units - changing, [65](#)
  - display units - definition, [65](#)
  - display units - guidelines, [66](#)
  - display units - querying, [66](#)
  - file name conflicts, [357](#)
  - file system, [355](#)
  - file types, [359](#)
  - HP-IB units - changing, [69](#)
  - HP-IB units - definition, [68](#)
  - HP-IB units - guidelines, [70](#)
  - HP-IB units - querying, [69](#)
  - IEEE 488.1 Interface Function Capabilities, [36](#)
  - IEEE 488.1 Remote Interface Message Capabilities, [38](#)
  - initializing (see Instrument Initialization), [326](#)
  - instruments contained in, [19](#)
  - interfacing to using serial ports, [380](#)
  - local mode, [44, 45](#)
  - operating modes, [18](#)
  - overview, [18](#)
  - remote mode, [44, 45](#)
  - remote operation, [35](#)
  - STATe command - definition, [75](#)
  - STATe command - guidelines, [76](#)
  - status registers, [280](#)
  - units of measure, [64](#)
  - writing programs for, [18, 22](#)
- TESTS Subsystem, [441](#)
  - default mass storage locations, [353](#)
  - DOS file restrictions, [360](#)
  - file descriptions, [442](#)
  - file relationships, [443](#)
  - screens, [444](#)
  - writing programs for, [441](#)
- TestSet
  - file name entry field width, [356](#)
  - file names (see also DOS & LIF file names), [356](#)
- Trigger
  - HP-IB commands, [259](#)
- Trigger - aborting, [260](#)
- Trigger event, [256](#)
- Trigger modes, [257, 260](#)
  - affect on measurement speed, [260](#)
  - default settings, [258](#)
  - Local/Remote Triggering Changes, [259](#)
  - retriggering, [257, 260](#)
  - settings for fastest measurements, [261](#)
  - settings for most reliable measurements, [261](#)
  - settling, [258, 260](#)
  - Triggering measurements, [256](#)

---

## U

- Uploading programs from Test Set to external controller, [439](#)
- Uploading programs from Test Set to PC, [415](#)
- Uploading programs to Test Set, [401](#)

## V

- Volume copy, [368](#)

## W

- Wildcards, [369](#)
- Word processor, [407](#)
  - configuring for program development, [407](#)
  - transferring programs to Test Set, [408](#)
  - writing lines of IBASIC code, [408](#)
- Write-protect switch, [363](#)

---

## X

Xon/Xoff, 386